(19)日本国特許庁（ＪＰ）　(12)特　許　公　報（Ｂ２）　(11)特許番号

第2915826号

(45)発行日　平成11年(1999)７月５日　(24)登録日　平成11年(1999)４月16日

(51)Int.Cl.⁶	識別記号	ＦＩ	
Ｇ０６Ｆ　17/50		Ｇ０６Ｆ　15/60	６２８Ａ
Ｇ０５Ｂ　19/19		Ｇ０５Ｂ　19/19	Ｍ
Ｇ０６Ｔ　17/00		Ｇ０６Ｆ　15/62	３５０Ａ

請求項の数５(全 51 頁)

(21)出願番号　特願平7－174627

(22)出願日　平成７年(1995)７月11日

(65)公開番号　特開平9－27046

(43)公開日　平成９年(1997)１月28日

審査請求日　平成９年(1997)１月24日

(73)特許権者　000005223
富士通株式会社
神奈川県川崎市中原区上小田中４丁目１番１号

(72)発明者　佐藤　裕一
神奈川県川崎市中原区上小田中1015番地
富士通株式会社内

(72)発明者　平田　光徳
神奈川県川崎市中原区上小田中1015番地
富士通株式会社内

(72)発明者　丸山　次人
神奈川県川崎市中原区上小田中1015番地
富士通株式会社内

(74)代理人　弁理士　斉藤　千幹

審査官　田中　幸雄

最終頁に続く

(54)【発明の名称】　干渉チェック装置

1

(57)【特許請求の範囲】

【請求項１】　非凸多面体と他の物体間の干渉チェックを行う干渉チェック装置において、

非凸多面体を構成する頂点を、第１軸（例えばＸ軸）の座標値の差が零あるいは微小量ε以内の複数の頂点群に分割する手段、

各頂点群について第２－第３軸平面（例えばＹＺ平面）上で２次元凸包を生成する手段、

隣接する２次元凸包を併合して３次元凸包を生成し、以後、隣接する３次元凸包を順次併合して目的とする非凸多面体の凸包を生成する手段、

該凸包と他の物体間の干渉チェックを行う手段、

凸包と他の物体が干渉し始めた時、凸包を解除して前記非凸多面体と他の物体間の干渉チェックを行う手段を備えたことを特徴とする干渉チェック装置。

2

【請求項２】　前記非凸多面体の凸包を生成する手段は、

隣接する３次元凸包の頂点間を接続する稜線のうち最も上に位置するアッパーブリッジを求める手段、

該アッパーブリッジを１辺とする最上位位置の三角形ポリゴンを求め、該ポリゴンで隣接３次元凸包間をラッピングし、該三角形ポリゴンに隣接する最上位位置の三角形ポリゴンを求め、該ポリゴンで隣接３次元凸包間をラッピングし、以後同様にラッピングを行って３次元凸包を併合する手段、

を備えることを特徴とする請求項１記載の干渉チェック装置。

【請求項３】　前記２次元凸包を生成する手段は、

第２軸（例えばＹ軸）の座標値の差が微小量以内の複数の頂点群に分割し、各頂点群毎に、第３軸（例えばＺ

軸）座標値の大きさ順に頂点をリンクして１次元凸包を生成し、隣接する１次元凸包を併合して２次元凸包を生成し、以後、隣接する２次元凸包を順次併合して目的とする２次元凸包を生成することを特徴とする請求項１記載の干渉チェック装置。

【請求項４】　前記２次元凸包を生成する手段は、前記隣接する２次元凸包を併合する際、隣接する２次元凸包の頂点間を接続する稜線のうち最も上に位置するアッパーブリッジと最も下に位置するロワーブリッジを求め、これら各ブリッジにより隣接する２つの２次元凸包を結合して新たな２次元凸包を生成することを特徴とする請求項３記載の干渉チェック装置。

【請求項５】　非凸多面体の集合における干渉チェック装置において、

各非凸多面体を包絡する球を構成し、各球を階層球で順次包絡し、階層包絡球のメタツリー構造を生成する手段、

前記生成されたメタツリー構造に基づいて、上位階層の包絡球同士の干渉チェックを行い、干渉する上位階層の包絡球を、該包絡球を構成する下位階層の包絡球に分解し、下位階層の包絡球について干渉チェックを行い、以後干渉しなくなるまで前記包絡球の分解及び干渉チェック処理を行って、近接球のペアを求める手段、

該近接球のペアに応じた近接非凸多面体のペアを求める手段、

該近接非凸多面体のペアについてそれぞれ凸包を生成する手段、

凸包同士の干渉チェックを行う手段、

干渉し始めた時、凸包を解除して前記非凸多面体間の干渉チェックを行う手段を備えたことを特徴とする干渉チェック装置。

【発明の詳細な説明】

【０００１】

【産業上の利用分野】本発明は非凸多面体である２つの物体間の干渉チェック方法に係わり、特に、コンピュータ上に構築されたＣＧモデルに対してそれらが互いに離れているか、干渉しているかを計算機上にて判定し、物体間の最近点、干渉点、最近点距離をリアルタイムに算出する干渉チェック装置に関する。本発明は、機構設計用ＣＡＤシステム、マニピュレータや自走車等の移動ロボットのパス生成、マルチメディアにおけるアニメーション作成、ゲームソフト等、コンピュータグラフィックスを応用した様々な分野に適用できるものである。

【０００２】

【従来の技術】グラフィックコンピュータ上に物体のＣＧモデル（物体形状を定めるポリゴン集合）が複数個、構築されている場合、任意の凹凸を持ったＣＧモデルが互いに離れているか干渉しているかを判定し、離れている場合には物体間の最近点を、干渉している場合には干渉点を、リアルタイムに算出することが望まれている。

尚、一般に、グラフィックコンピュータ上のＣＧモデルに限らず、物体の数学モデル（物体を定義する３次元頂点座標、稜線集合、面集合）が与えられた時、数学モデル間の最近点／干渉点を求める問題を「最近点探索問題」という。かかる最近点探索問題に対する基本的なアプローチは、対象となる物体の形状、位置、姿勢データを計算機上に取り込み、初等幾何学的計算により解決する方法である。通常、ＣＡＤ分野で使われているグラフィックシュミレータでは、平面的な凸多角形（凸ポリゴン）を複数枚張りつけて一つの物体を表現している。従って、二つの物体間の干渉状態をチェックする最も素朴な方法は、各々の物体を構成しているポリゴン間の干渉チェックを全てのポリゴンの組合せについて行うものである。具体的には、空間上に任意に配置された二つのポリゴンに対し、最も近接しているポイントを各格子点、辺、面について探索し、最接近距離を算出することである。勿論、最接近距離がゼロならば、二つのポリゴン間は干渉状態にあるということである。この方法では、二つの物体を表現している格子点数をM_1，M_2としたときにＯ（$M_1 \cdot M_2$）の計算負荷を伴い、リアルタイム性が失われていた。

【０００３】一方、歴史的には、各格子点間の垂直二等分線を結んで各格子点の近接関係を図示するVoronoi図を使った方法や、三次元空間を象限ごとに次々に八等分して対象の専有領域をデータベース化するオクトツリーの方法による最近点の探索法も知られているが、これらの方法は何れも対象が静的に一定である場合に有効であり、分解や組み立てなどにより対象が動的に次々と変化していく場合には不向きであった。ロボテックスの分野では、Boblow法、Lin/Canny法、Gilbert法が高速な干渉チェックアルゴリズムとして知られており、何れもＯ（Ｎ）（Ｎ：対象の総格子点数）以下の計算時間で最近点を算出することができる。しかしながら、これらの方法を適用できる対象は何れも凸多面体（Gilbert法では楕円体などの二次凸曲面を持った物体にも適用可能）に限られている。

【０００４】Pro/Engineer等の市販の三次元ＣＡＤシステムを使って設計される形状データは、殆どの場合において非凸物体であり、そのため、部品間の干渉チェックを行うのに、Boblow法、Lin/Canny法、Gilbert法等のリアルタイム性を有する方法は適用されず、もっぱら、部品を構成するポリゴンに関して総当たり的に干渉チェックを行う方法(総当たり法)や形状を定義する方程式を解析的に時間を掛けて解く方法が取られていた。非凸多面体間の干渉チェックとしては、多面体を階層的に並べた球の集合で覆い、球間の干渉チェックを行うことにより最も接近しているポリゴン間の組合わせを求め、Gilbert法を適用するQuinlan法が知られている。この方法は、任意の非凸多面体に適用できる極めて一般的方法であるが、厳密に２物体間の距離を求めようとすると探索空間

が増え、計算時間が極端に長くなる欠点を有している。

【０００５】

【発明が解決しようとする課題】従来の干渉チェック方法では、リアルタイムに非凸多面体が互いに干渉しているかを判定し、離れている場合には物体間の最近点を、干渉している場合には干渉点を算出できない問題があった。以上から本発明の目的は、高速に非凸多面体間の干渉チェックを行うことである。本発明の別の目的は、干渉チェックの前処理である凸包の生成を効率良く、高速に行って干渉チェックすることである。

【０００６】

【課題を解決するための手段】図１は本発明の原理説明図である。１は非凸多面体の形状データより非凸多面体を包絡する最小の凸多面体（凸包:convex hull)を生成する凸包構成部、２は凸包の近接点線形リストを作成する近接点線形リスト作成部、３は非凸多面体を構成するポリゴン上に所定半径の複数の球（リーフ球）を配列して各ポリゴンを覆い、該球を階層球で順次包絡し、階層包絡球の２分木ツリーを生成する階層包絡球２分木ツリー生成部、４は生成された凸包、近接点線形リスト、２分木ツリーのデータを記憶する記憶部、５はGilbert法等に基づいて凸包間の干渉チェックを行う凸包間干渉チェック部、６は２分岐ツリー構造に基づいて、上位階層の包絡球同士の干渉チェックを行い、干渉する上位階層の包絡球を、該包絡球を構成する下位階層の包絡球に分解し、下位階層の包絡球について干渉チェックを行い、以後干渉しなくなるまで前記包絡球の分解及び干渉チェック処理を行って、近接球のペアを求め、該近接球のペアに応じたポリゴンのペアを近接ポリゴンのペアとして求めるバブルコライジョン法実行部、７は近接ポリゴンペア間の干渉チェックを行うポリゴンペア間干渉チェック部である。

【０００７】

【作用】

(a) 干渉チェック高速前処理

非凸多面体の干渉チェックを高速に実行するには、凸包や近接点線形リストの生成、階層球による物体の包絡のための前処理を実行する必要がある。本発明においては、凸包構成部１、近接点線形リスト作成部２、階層球２分木ツリー生成部３が干渉チェック高速前処理により、それぞれ凸包、近接点線型リスト、階層球２分木ツリーを生成する。これにより、頂点数Ｎ個の任意の非凸多面体ＣＧモデルに対して、Ｎlog Ｎの時間で凸包、凸包間連続探索のための近接点線形リストを生成でき、更に、バブルコライジョン法(Bubble Collision法)のための「階層球による物体の包絡」を行うことができる。

【０００８】1) 干渉チェック高速前処理（凸包構成）：非凸多面体の凸包を生成し、該凸包と他の物体間の干渉チェックを行い、干渉し始めた時、凸包を解除して前記非凸多面体と他の物体間の干渉チェックを行う干渉チェック方法において、凸包構成部１は、非凸多面体を構成する頂点を、第１軸（例えばＸ軸）の座標値の差が零あるいは微小量ε以内の複数の頂点群に分割し、各頂点群について第２－第３軸平面（例えばＹＺ平面）上で２次元凸包を生成し、隣接する２次元凸包を併合して３次元凸包を生成し、以後、隣接する３次元凸包を順次併合して目的とする非凸多面体の凸包を生成する。

【０００９】2) 干渉チェック前処理（近接点線形リスト）：この前処理は、凸多面体同士の干渉チェックにおいて、最近点を連続的に探索していくための前処理であり、この処理を施しておくことにより最近点の連続探索が極めて高速化される。近接点線形リストとは、ＣＧモデルを構成する各頂点に対し、その頂点と稜線を介して繋がっている他の頂点とを線形リストで繋ぐデータ構造のことである。リストの先頭から順次辿ることにより、その頂点に繋がる全ての近接点を拾いだすことができる。すなわち、近接点線形リスト作成部２は、凸包を構成する全三角ポリゴンの頂点を第１方向（next方向）にリンクし、各頂点にポリゴン辺を介して繋がる頂点群を第２の方向（branch方向）に該頂点にリンクしてなるデータ構造を備えた近接点線形リストを作成して出力する。本前処理は、凸包構成の処理に引き続いて行われるところに特徴がある。

【００１０】3) 干渉チェック高速前処理(近接点線形リスト結果のダンプ、およびそれらの高速ローディング)：近接点線形リストの結果をダンプし、それらのダンプファイルをメモリ上に高速ローディングする。前処理を施していないＣＧモデルに対しては、凸分解、近接点形リスト生成処理を施すが、一旦、近接点線形リストを表現するファイルを作成した後は、このファイルから高速ローディングを行ってメモリ上に展開する。これにより、Ｎポリゴンからなるモデルに対して、Ｏ（Ｎ）の時間で近接点線形リストをメモリ上にの展開することが可能となる。本処理は、凸包構成および近接線形リスト構成の処理に引き続いて行われるところに特徴がある。

【００１１】4) 干渉チェック前処理（階層球による物体の包絡）：非凸多面体を構成する各ポリゴンを、該ポリゴン上に中心を持つ複数の球の組み合わせで覆い、それらの球を包含する球を階層的に構成する。これにより、球間のチェックを階層的に行い最も接近した近接球に応じたポリゴンの組み合せを高速に探索することが可能となる。階層球の構成は、分割統治法で行われ、ＮポリゴンからなるＣＧモデルに対して、Ｏ（ＮLog Ｎ）の時間で処理が終わる。すなわち、階層球２分木ツリー生成部３は、非凸多面体を構成するポリゴン上に所定半径の複数の球（リーフ球）を配列して各ポリゴンを覆い、分割統治アルゴリズムに従って該球を階層球で順次包絡し、階層包絡球の２分木ツリー構造を生成する。この場合、リーフ球の半径値を、非凸多面体を包含する球の半

径に応じた値にすると共に、非凸多面体のポリゴン数が少ない場合には小さくし、ポリゴン数が多い場合には大きくする。本処理とQuinlan法における階層球の構成法の違いは、Quinlan法の階層球構成法がscan conversionに類似した方法であるのに対し、本手法は、対象を覆うポリゴン数を考慮し、スケールＳの値をパラメータとした効率的な球構成を行う。

【００１２】5) 非凸多面体干渉チェックにおけるBubble Collision法：非凸多面体間の干渉チェックを高速に行うために、階層球２分木ツリー生成部３は各々の非凸多面体を階層球で包絡し、包絡球からなるツリーを構成する。バブルコライジョン実行部６は、球間の干渉チェックをツリーに基づいてDepth-firstの探索法でチェックしていき、最も接近する球のペアを検索する。近接ポリゴン間干渉チェック部７は、検索した球の中心が載っているポリゴンの組み合わせに対してGilbert法を適用することにより、二つの非凸多面体間の最近点を算出する。この場合、バブルコライジョン実行部６は、球間の距離を以下により計算する。すなわち、バブルコライジョン実行部６は、リーフ球とリーフ球間の距離をこれらリーフ球の中心間距離とし、リーフ球と非リーフ球間の距離をこれら球の中心間距離から非リーフ球の半径を差し引いた距離とし、非リーフ球と非リーフ球間の距離をこれら球の中心間距離から両非リーフ球の半径を差し引いた距離として計算する。本手法Quinlan法との違いは、球間の距離の計算法にある。本手法では、球間の距離を求めるに際し、その球が包絡球ツリーのどの深さにあるかによって、球間の距離の定義を変更する。これにより、Bubble Collision法で問題となっている、最終的にチェックしなければならないポリゴンペアの数を抑えることができる。

【００１３】(b) Bubble collision法との融合

1) 凸多面体干渉チェックにおける凸包間連続型Gilbert法と非凸多面体用Bubble Collision法の融合：非凸多面体間の干渉チェックを出来るだけ無駄なく行うために、非凸多面体間が充分離れている場合には、非凸多面体を包含する凸包を構成し、凸包間に連続型Gilbert法を適用する。凸包同士が干渉を始めたならば、Bubble Collision法に自動的にスイッチする。スイッチングの指標は、凸包間の距離とする。すなわち、凸包構成部１により非凸多面体の凸包を生成し、非凸多面体間の距離が十分離れている場合には凸包間干渉チェック部５により凸包間の干渉チェックを行い、凸包同士が干渉し始めたら、バブルコライジョン実行部６によりBubble collision法を適用して非凸多面体間の干渉チェックを行う。かかる凸包間連続探索法と非凸多面体間の干渉チェックのためのBubble Collision法の融合(連続型 Bubble Collision法)により、最近点を求めるに当たり、無駄な計算が極力省かれる。

【００１４】2) ４次元Gilbert法：凸包間連続探索法から非凸多面体用Bubble Collision法へスイッチングする場合、またはその逆において、スイッチングの指標を凸包間連続型Gilbert法により算出した距離である。凸包間が干渉しだしたならば、距離を算出するアルゴリズムを３次元連続型Gilbert法から４次元連続型Gilbert法にスイッチングし、一方の凸包が他方の凸包に吸収された場合においても矛盾無くスムースにスイッチングが行われる。

3) メタツリーによる高速化：非凸多面体の集合において、最も近接する非凸多面体のペアを高速に算出するために、非凸多面体を包絡する球を構成し、この球を包含する包絡球を階層的に構成することによるメタツリーを構成する。非凸多面体間を総当たり的にチェックする代わりに、メタツリーに基づいて包絡球間をdepth-firstの探索法でチェックしていき、最も接近する球のペアを検索する。検索した球の中心が載る凸包間に連続型Gilbert法を適用することにより、最近点を効率よく算出する。本手法は、メタツリーの構成法、メタツリー間の干渉チェックから連続型Gilbert法およびBubble Collision法に移行する手法に特徴がある。

【００１５】4) メタツリー構成を含めた連続干渉チェック法：任意の非凸多面体間の最近点を連続的に追跡していく場合において、メタツリー構成、Gilbert法、Bubble Collision法の全てを数サイクルに一回のみ行い、他のサイクルでは、前回のサイクルで使用したポリゴンペアに再度Gilbert法を適用することにより最短距離を求め、計算負荷を大幅に軽減する。

5) サイクルタイム自動調整型連続干渉チェック法：連続干渉チェック法において、Bubble Collision法に掛かる計算時間を自動測定することにより、メタツリー構成、Gilbert法、Bubble Collision法を行うサイクルを自動的に決定する。

【００１６】(c) 連続型Bubble Collision法による干渉チェック：

1) 連続型Bubble Collision法

干渉チェックにおいては、凸包同士の干渉チェックにおいて凸包同士が干渉すると、凸包内部での干渉チェックをBubble Collision法により行う。Bubble Collision法による干渉チェックは時間を要するため、バブルコライジョン実行部６は、毎サイクルBubble Collision法を実行せず数サイクルに一度行い、Bubble Collision法の実行結果である近接ポリゴンの幾つかのペアを保存する。移動距離／姿勢変化が小さいとすると、次の瞬間もそのペアは同じであるので、その数組の近接ポリゴン間に対してのみ、近接ポリゴン間干渉チェック部７は干渉チェックを行い、最近点、干渉点を求めて出力する。保存した数ペアのみに関して干渉チェックを行うことにより、時間を要するBubble Collision法の処理回数を減少でき、計算時間の短縮が図られる。

【００１７】2) Bubble Collision法 の計算時間による

連続型Bubble Collision法の更新サイクル自動決定法：連続型Bubble Collision法においては、物体形状と移動速度に基づいて、近接ポリゴンペアの更新サイクル（次のBubble Collision法実行までのサイクル）の調整が必要である。このサイクルが長すぎると、修正が遅くなり、間違った結果を出力するのが目立つ。一方、上記更新サイクルが短いと、リアルタイム性が薄れる。このため、試行錯誤的なサイクルの調整は、難しい。そこで、Bubble Collision法を行う毎にその計算時間を測定し、その計算時間から次の更新サイクルを自動的に決定する。

【００１８】3) 連続型Bubble Collision法の修正方法（物体移動量による判定）：連続型Bubble Collision法は移動距離／姿勢変化が小さいと仮定するので、大移動が発生すると、頻繁に間違った結果を出力するのが目立つ。移動距離／姿勢変化があるしきい値より大きい場合には、バブルコライジョン実行部６は、近接ポリゴンペアの更新サイクルに達しなくてもBubble Collision法を行い、近接ポリゴンペアを更新する。これにより、正しい干渉チェック結果が得られる。

4) 連続型Bubble Collision法の修正方法（物体移動速度による判定）：連続型Bubble Collision法は移動距離／姿勢変化が小さいと仮定するので、大移動が発生すると、頻繁に間違った結果を出力するのが目立つ。移動速度があるしきい値より大きい場合には、近接ポリゴンペアの更新サイクルに達しなくてもバブルコライジョン実行部６はBubble Collision法を行い、近接ポリゴンペアを更新する。これにより、正しい干渉チェック結果が得られる。

【００１９】5) 連続型Bubble Collision法の修正方法（最近点距離比による判定）：連続型Bubble Collision法は移動距離／姿勢変化が小さいと仮定するので、大移動が発生すると、頻繁に間違った結果を出力するのが目立つ。最近点距離比があるしきい値より大きい場合には、近接ポリゴンペアの更新サイクルに達しなくてもバブルコライジョン実行部６はBubble Collision法を行い、近接ポリゴンペアを更新する。これにより、正しい干渉チェック結果が得られる。

6) 凸包外における物体静止時の干渉チェックの効率化：干渉チェックを一度行った後に、チェック対象物体がすべて静止していれば、前サイクルの結果を更新する必要はない。速度指令、位置変位が零ならば、前サイクルの干渉チェック結果をそのまま使って干渉チェックの計算を省く。この方法によれば、干渉チェック対象物体グループが二つ以上存在する場合、移動物体を含むグループのみの干渉チェック計算をするだけで良いので効果が大きい。

【００２０】7) 凸包内における物体静止時の干渉チェックの効率化:凸包内では連続型Bubble Collision法を行っているが、干渉チェックを一度行った後に、チェック対象物体がすべて静止していれば、前サイクルの結果を更新する必要はない。速度指令、位置変位が零ならば、前サイクルの干渉チェック結果をそのまま使って干渉チェックの計算を省く。但し、静止後の近接ポリゴンペアの更新サイクルにおいて一度だけBubble Collision法を行なって、正しい近接ポリゴンペアを求める必要がある。この方法によれば、干渉チェック対象物体グループが二つ以上存在する場合、移動物体を含むグループのみの干渉チェック計算をするだけで良いので効果が大きい。

【００２１】

【実施例】

（Ａ）凸包の生成

（a） 非凸多面体の干渉チェックの概略

非凸多面体間の干渉チェックを行う方法として、非凸多面体（図２)を包絡する最小の凸多面体（凸包：convex hull)を生成し（図３）、非凸多面体間の距離が十分離れている場合には凸包間の干渉チェックを行い、凸包同士が干渉し始めたら、Bubble collision法を適用して非凸多面体間の干渉チェックを行う。かかる非凸多面体間の干渉チェックを行うためには、前処理により非凸多面体より凸包を生成する必要がある。

【００２２】(b) システム構成

図４は凸包生成の前処理や干渉チェック処理を実行するシステム構成図であり、１１はグラフィックワークステーションＧＷＳ、１２は三次元ＣＡＤシステムである。ワークステーション１１において、１１ａは干渉チェックの前処理や干渉チェック処理、その他の処理を実行するプロセッサ、１１ｂはＲＯＭであり、凸包生成の前処理プログラムＣＨＰ、干渉チェックプログラムＩＣＰ等を記憶するもの、１１ｃはＲＡＭであり、三次元ＣＡＤシステム１２で設計された形状データ、その他の設定データ、演算処理結果を記憶するもの、１１ｄはＣＡＤシステムとの通信インタフェース、１１ｅは表示部、１１ｆはキーボード、１１ｇはマウス等の操作部、１１ｈはフロッピーディスク用のディスクドライブ、１１ｉはハードディスクである。

【００２３】(c) 概略フロー

図５は凸包生成の概略処理（干渉チェック前処理）フローである。三次元ＣＡＤシステム１２は形状モデル（非凸多面体）を作成してグラフィックワークステーション１１に入力し、ＲＡＭ１１ｃに記憶する（ステップ１００１、１００２）。ＣＡＤシステム１２は、形状モデルデータとして非凸多面体をポリゴンで覆った場合における全ポリゴンデータを入力する。ポリゴンデータは、ポリゴンの各頂点の座標値とその点における光の反射方向を示す情報を有している。最も一般的なポリゴンデータは三角形ポリゴンの張り合わせで表現されるものである。三角形ポリゴンデータにおいて、normal に続く３つの実数が光の反射方向を示し、vertex に続く３つの

実数が頂点座標値を表している。

【００２４】ついで、前処理（凸包生成）の実行を指令すると、プロセッサ１１ａはＲＯＭ１１ｂに記憶されている凸包生成用のプログラムＣＨＰに基づいて非凸多面体の形状データを用いて凸包を生成する（ステップ１００３）。凸包生成後、近接点線形リスト構成アルゴリズムを用いて後述する近接点線形リストを作成してＲＡＭ１１ｃに展開する（ステップ１００４，１００５）。しかる後、該近接点線形リストデータに基づき専用ダンプファイル～．ＨＬを作成し、ハードディスク１１ｉ等に記憶する（ステップ１００６）。以後、該近接点線形リストが必要な場合には、新たに作成せず近接点線形リスト展開アルゴリズムを通じて専用ダンプファイル内の情報をデータ形式に再現する（ステップ１００７）。

【００２５】(d) 多面体表現のためのデータ構造

多面体は三角形パッチで張られているとする。多面体の各三角形パッチ（図６(a)参照）をＣ言語で表現すると図７に示す形式のデータ構造を取る。図７において、最も基本的は表現クラスは、頂点Vertexではなくエッジ Edgeである。各エッジは方向を持っており、頂点ｕからｖに向かうエッジを指すポインタ変数をｅ、原点ｕを指すポインタをoriq、対称エッジを指すポインタをsym、頂点ｖから頂点ｗに向かうエッジを指すポインタをlnext、頂点ｕから頂点ｗに向かうエッジを指すポインタをonextとする。整数変数colorは、凸包構成の過程で必要となり、blue（青）、purple（紫）、red（赤）のどれかの値を取る。blue、purple、redの意味は後述する。

また、三角形ｕ－ｖ－ｗを表現するクラスとしてFacet＊を用意し、各エッジは左手方向のFacetを指すポインタ変数leftを持つ。

【００２６】頂点を表すクラスVertexは、頂点座標を表す３次元ベクトルｖ及びベクトルｖにランダムな偏差をのせたｄｖを持ち(Vector３は３次元実ベクトルを表す型名）、更に輪環構造を表現するためのポインタ変数next,preを持つ。Convexは、凸包を表現するためのクラスであり、凸包を構成するエッジの先頭要素を指すポインタ変数としてedgeを持つ。図６(b)で示すとおり、edge->oriq,edgemin->oriqはそれぞれ凸包Ｈの Ｘ軸方向に対する最大頂点、最小頂点を指すポインタ変数である。頂点を指すポインタ変数 maxX, minX, maxY, minY は、凸包構成の前半の過程でのみ使用する。また、sideには分割統治法における左右の区別、vcntには凸包を張る全頂点数が入る。ポインタ変数 vertexesはedgemin->oriqと同じ頂点を指し、vertexesから出発してnext方向に辿ることにより凸包を構成する全ての頂点を小さい順に探索できる。但し、この頂点探索は、図６(b)のV_1，V_t，V_kのような凸包内部の頂点にもわたるものとする。vertexesは、分割統治法の前半およびバックアップ処理にて使用する。Listは、凸包構成の最後の過程で必要となるクラスであり、詳細は後述する。

【００２７】図７のデータ構造により、ある一つの頂点に対して、その頂点と稜線を介して繋がる頂点を時計回り、反時計回りに取り出す操作は、以下の手続で与えられる。尚、以後のプログラムはすべてＣ言語で表現したものである。

```
Edge *edumy0 = sym;
Edge *edumy1 = sym;
     .
     .
edumy0 = edumy0->lnext->sym;        //  clockwise order
     .
     .
edumy1 = edumy1->sym->onext->sym;   //  counterclockwise order
```

【００２８】(e)３次元凸包構成アルゴリズム

図８は、図５のステップ１００３における３次元凸包構成アルゴリズムであり、凸包構成の基本関数を示している。本アルゴリズムを一言で言えば、Ｘ軸方向に対して各頂点をソーティングしながら分割統治法を行うものである。詳細は、以下の手続からなる。三角形ポインタの頂点データを入力する（ステップ１１０１）。入力形式は、ステップ１１０１のブロック内に示すようにクラスConvexのポインタ変数conのメンバ変数con->vertexesを先頭要素としてnext,pre方向にvertexを繋げ、輪環形式を取る。vertexesに繋がるvertexは、図７の入力ファイルの頂点データを入力順に繋げたものであり、Ｘ軸方向のソーティングは行っていない。ついで、モデルデータconに対し、Ｘ軸方向のmax，minであるmaxX，minXを算出する（ステップ１１０２）。すなわち、頂点集合vertexesに対してＸ軸方向の最大値、最小値を取る頂点を検索し、各々のアドレスを maxX， minXに入れる。

【００２９】しかる後、Ｘ座標値の最大値、最小値の差が微少量εよりも小さいか調べ、小さい時は２次元凸包を構成する(ステップ１１０３）。すなわち、頂点集合vertexesがＸ軸に垂直な平面上に載っており、Ｘ座標値の最大値maxX->v(0)、最小値minX->v(0)の差が微少量εよりも小さい時は、２次元凸包構成関数 MakeConvex2Dim(con)を作用させて2次元凸包を生成する。Ｘ座標値の最大値、最小値の差が微少量εよりも小さくなければ、頂点集合vertexesをＸ軸方向に２分割する（ステップ１１０４）。すなわち、最大値maxX->v(0)と、最小値minX->v(0)の中心値を境にして、頂点集合vertexesをconLef

t,conRightに２分割する。例えば、ｓ=(maxX->v(0)+minX->v(0))/2とする時（ｓは中心値）、vertex->v(0)<sならばconLeft->vertexesに繋げ、vertex->v(0)≧ｓならば conRight->vertexesに繋げる。

【００３０】ついで、左右二つに分割したConvexクラス、conLeft, conRightに対して再帰的に凸包構成関数MakeConvex( )を作用させて凸包を生成する（ステップ１１０５）。以後、２分割ができなくなるまで凸包 conLeft, conRightのマージング(併合)を行う(ステップ １１０６）。すなわち、再帰的に構成された凸包conLeft, conRightを併合して一つの凸包を構成する。一般に、分割統治法は、上記手続きに示す通り一つの集合を再帰的に次々に二分割していき併合の過程を繰り返すことにより最終的な出力を得る。分割統治法では、そのアルゴリズム依存性は主にマージプロセス内で表現される。図８の手続きでは関数 Merge3Dim(conLeft,conRight)がマージプロセスを表現している。Merge3Dim(conLeft,conRight)の詳細は、後述する。

【００３１】(f) ２次元凸包構成アルゴリズム

２次元凸包構成の関数は、図９に示す通り、３次元の場合と殆ど同じ構成をしている。違いは、ステップ１２０１の１次元凸包構成のプロセスとステップ１２０２の２次元におけるマージプロセスである。ステップ１２０１の１次元凸包構成のプロセスは図１０に示す通り、Ｚ軸に平行に並んだ頂点のトップとボトムを互いにnext,preで結ぶことで終了する。トップとボトムが同一点の場合には、next,preは同じ頂点を差し示す。図１１は、ステップ１２０２における２次元におけるマージプロセスMerge2Dim(conLeft,conRight)の詳細を示したものである。図に示す通りMerge2Dim(,)は、凸包conLeft、conRightを上方から覆う稜線 Upper Bridgeと下方から覆う稜線Lower Bridgeを見つけるプロセスである。Upper Bridgeをみつけるために、econLeft->maxY、econRight->minYを結ぶ稜線を初期値として、conLeft、conRightに対して各々next,preの方向に順次移行して行き、最も上方に達したところの頂点を結んでUpper Bridgeとする。同様に、econLeft,econRightに対して各々pre,next方向に移行していくことによりLower Bridgeを見つけることができる。

【００３２】図１２は、マージ関数 Merge2Dim(conLeft,conRight)の詳細構成を示したものである。ステップ１３０１で初期値を設定し、ステップ１３０２のループ(do～while文)でUpper Bridgeを、ステップ１３０３のループ(do～while文)でLower Bridgeを見つけ出す。W->ConvexJudgeUp2Dim(u,v)はｕ，ｖを結ぶ直線に対し、頂点wの上下関係を判定する関数である。

【００３３】図１３Ａ～図１６は凸包構成の説明図である。図１３において、２１は頂点ａ～ｋを有する非凸多面体であり、頂点ａ～ｅのＸ座標値は等しくＸ＝x_1、頂点ｆのＸ座標値はＸ＝x_2、頂点ｇ～ｋのＸ座標値は

Ｘ＝０である。図１３の非凸多面体に対して、図８の三次元凸包構成アルゴリズムを適用すると、各頂点ａ～ｋはＸ座標値の差が微小量εの範囲内の第１頂点群ｇ～ｋ（図１４(a)参照)、第２頂点群ａ～ｅ（図１４(b)参照)、第３頂点群ｆ（図１４(c)参照)に分けられる。これら各頂点群に対して二次元凸包構成アルゴリズムを施す。例えば、第２頂点群ａ～ｅに２次元凸包構成アルゴリズムを施すと、Ｙ座標値の差がεの範囲内の第１頂点群ａ，ｂ、第２頂点群ｄ、第３頂点群ｃ，ｅの３つに分けられ、これら各群に１次元凸包構成アルゴリズムを施す。しかる後、１次元凸包に対して２次元凸包構成のためのマージ関数 Merge2Dim(,)を適用して２次元凸包を構成する。例えば、第２、第３頂点群より生成した１次元凸包にマージ関数を適用すると、図１５(a)に示すように稜線ｄｅがupper bridge、稜線ｄｃがlower bridgeになり、２次元凸包ｃｄｅが生成される（図１５(b))。ついで、第１頂点群ａ，ｂにより生成した１次元凸包と２次元凸包ｃｄｅにマージ関数を適用すると、図１５(c)に示すように、稜線ａｅがupper bridge、稜線ｂｃがlower bridgeになり、２次元凸包ａｂｃｅが生成される（図１５(d)参照)。

【００３４】同様に、第１頂点群ｇ～ｋに対して２次元凸包構成アルゴリズムを施して２次元凸包を生成する。以上の２次元凸包の生成が終了して、３次元凸包構成用のMerge3Dim(,)を適用すると、３次元凸包が生成できる。例えば、２次元凸包ａｂｃｅと２次元凸包ｆにMerge3Dim(,)を適用すると、３次元凸包ａｂｃｅｆが生成され（図１６(a))、この３次元凸包ａｂｃｅｆと２次元凸包ｇｈｉｋにMerge3Dim(,)を適用すると三次元凸包ａｂｃｅｆｇｈｉｋが生成される（図１６(b))。３次元凸包構成アルゴリズムの本質は、図１７に示すマージ関数に集約され、以下の四つのフェーズからなる。

【００３５】Phase 0：頂点数が少ない場合における凸包の直接構成

凸包の直接構成における注意点は、頂点数が３以下の場合の取り扱いである。この場合には、最小凸包である四面体を張ることができないため、マージ過程を通じて頂点数を増やし、四面体を直接構成することが必要となる。本フェーズは、conLeft, conRightの頂点数が少ない場合におけるダイレクトな凸包構成プロセスであり、MakeConvexLowNumber(,), MakeConvexBackL(,),MakeConvexBackR(,)の各関数で表される。 MakeConvexLowNumber(,) は、内部においてconLeft->vcnt, conRight->vcntの値に応じて更に場合分けされる。conLeft、conRightの頂点数の合計が４未満の場合は、conLeft, conRightの頂点をnext,preのポインタで張り直し、vertexes上に蓄える。頂点数の合計が４になったところでMakeTetra22(,), MakeTetra13(,)の諸関数で四面体を直接構成する。conLeft->vcnt=2, conRight->vcnt=3の場合、conLeft->vcnt=3, conRight->vcnt=2の場合およびconLeft->v

cnt=3，conRight->vcnt=3の場合には、まず、四面体を直接構成し、後ほど説明するバックアッププロセス(図２８)を援用して凸包を構成する。

【００３６】MakeConvexBackL(,)は、conLeft->vcntの値が３以下の、conRight->vcntが４以上の場合のマージ法である。conRightが既に四面体以上の凸包を構成しているため、本関数は、凸包conRightにconLeftの頂点を一つ一つマージしていく過程を表す。本関数は、バックアッププロセスを使って構成される。同様に、MakeConvexBackR(,)はconRight->vcntが３以下、conLeft->vcntの値が４以上の場合のマージ関数である。以上、MakeConvexLowNumber(,)，MakeConvexBackL(,)，MakeConvexBackR(,)の処理により、phase1以降は、conLeft，conRightとも四面体以上の凸包を構成している場合の処理である。

【００３７】Phase 1：Upper Bridgeの構成

図１８に示すようなUpper Bridgeを構成する。この処理をできるだけ少ない計算回数で実行するために、conLeftのＸ軸方向最大エッジconLeft->edgeおよびconRightのＸ軸方向最小エッジconRight->edgeminを初期値にして近傍頂点を駈け上がっていくようにする。そのために、各頂点において、その近傍頂点をＸ－Ｚ平面に射影して２次元の凸包構成を行い、Ｘ－Ｚ平面上で近傍頂点と現在のBridgeの上下判定を繰り返していく。

【００３８】Phase 2：Wrapping process

３次元マージプロセスにおいて基幹となるものであり、その概要は、図１９に示すように凸包conLeft，conRightを包込むプロセスである。包み込む過程で、各凸包conLeft，conRightのエッジのcolorはBLUE(青)からPURPLE(紫)、またはRED(赤)に変化させていく。全てのエッジの初期colorはBLUEであるとし、図１９で示すようにWrappingの境界となるエッジのcolorはPURPLE，PURPLE edgeの近傍エッジで、併合した凸包の内部に入るエッジのcolorはREDに変化させる。他のエッジのcolorは全てBLUEのままである。最終的にマージングが完了した時点で併合凸包のエッジのcolorをBLUEにリセットする。

【００３９】Wrapping processの概要は以下の通りである。Upper Bridgeの頂点 utop，vtopの近傍頂点を検索し、Wrappingを構成する三角形パッチのもう一方の頂点となる候補を捜し出す。今、utopに対する候補頂点をFirstWinLとする時、utopの他の近傍頂点は三角形utop-vtop-FirstWinLが張る平面の下に位置する。一般に、FirstWinLは、Upper Bridgeから見て時計回り方向と反時計回り方向に２頂点存在するが、三角形をutop-FirstWinL-vtopの順に見た時に法線方向が上向きになる条件を課すことにより一意的に時計回りの頂点を取り出すことができる。同様に、vtopの近傍頂点を調べFirstWinRを取り出す。最後に三角形utop-FirstWinL-vtopの張る平面とFirstWinRとの上下関係を調べることにより、最初の三角形パッチの頂点を選択する。図１９は、FirstWinLが最初に選択された例を示している。FirstWinLが選択された後は、FirstWinL-vtopをUpper Bridgeと見做して同様の操作を繰り返す。図１９の例ではFirstWinLの近傍頂点を検索した結果、三角形 FirstWinL-u-vtopが選び出され、三角形 FirstWinL-u-vtopとFirstWinRの上下関係を調べ、FirstWinRが選択された例である。この場合は、三角形FirstWinL-FirstWinR-vtopが２番目の三角形パッチである。以後ｕ，ｖがutop，ｖtopと一致するまでｕ，ｖを更新していく。FirstWinL，ｕを一般的にwinning vertexと呼ぶ。

【００４０】２番目以降のwinning vertexの検索では、近傍頂点の全てに渡る検索は必要ない。直前の境界エッジを初期値として、conLeftの場合には時計回り、conRightの場合には反時計回りに検索することにより、候補頂点の一意的な選出が可能となる。winning vertexを選出していく過程で、境界エッジ、およびその近傍エッジのcolorを変化させていく。第１三角形パッチの場合には、境界エッジ、(図１９の例ではエッジutop-FirstWinL) のみのcolorをPURPLEに変化させる。第２三角形パッチ以降は、境界エッジをPURPLEに変えると共に、winning vertexの近傍エッジの内、併合凸包の内部に隠れるエッジのcolorを全てREDに変える。以後、この処理を最終三角形パッチまで繰返し、最後にutop，vtopの近傍エッジを調べ、内包するエッジのcolorをREDに変える。以上、本プロセスでは、併合凸包を構成する三角形パッチの頂点集合が全て、選別され、それらが、三角形パッチ毎に図７のクラスListに格納され、next方向に線形リストを作成する。

【００４１】Phase 3:三角形ポリゴンのPatching process

Phase2で構成した三角形パッチの頂点集合からなる線形リストListに基づき、実際に三角形ポリゴンを張つけていくプロセスである。本プロセスは、図２０、図２１に示す通り、大きく分けて初期パッチ、中間パッチ、最終パッチの三つの過程からなる。三つの過程における三角形パッチの構成法は微妙に違い、初期パッチでは、一本のエッジと一個の頂点からなる三角形パッチの構成法、中間パッチでは、２本のエッジからなる三角形パッチの構成法、最終パッチでは３本のエッジからなる三角形パッチの構成法が使用される。各々の構成法を表現する関数PatchTriangle(,)では、図６(a) に示すonext，lnext，sym，orig，leftの各ポインタ変数が、となり合う三角形パッチ間において矛盾がないように設定される。

【００４２】Phase 4：後処理

マージプロセス１サイクル毎の後処理を表し、以下の三つのプロセスからなっている。

①マージング後におけるconLeft,conRightの境界エッジのcolorをPURPLEからBLUEに変更する。

②併合凸包のＸ軸方向の最大エッジedge、最小エッジedgeminを選出する。

❸併合凸包のＸ軸方向の最大頂点maxX,最小頂点minXを選出する。

❷の過程で注意しなければならないのは、選び出したエッジのcolorがREDの可能性があることである。この場合には、このエッジを初期値として時計回りまたは反時計回りのエッジ検索を行いcolorがBLUEのエッジを探し出し、edge,edgeminとして設定する。❸の過程が必要な理由は、後ほど説明する。バックアッププロセスのためである。

【００４３】(h) Ｏ（Ｎlog Ｎ）の根拠

conLeftの頂点数をn_1、conRightの頂点数をn_2とする時、(g)のマージプロセスにかかる計算負荷はＯ（n_1+n_2）である。従って、総数Ｎの頂点集合に対する凸包構成の計算時間をＴ（ｎ）とすると、

Ｔ（Ｎ）＝２Ｔ（Ｎ／２）＋Ｏ（Ｎ）

という漸化式が得られる。これを解くと、Ｔ（Ｎ）＝Ｏ（ＮｌｏｇＮ）となる。

【００４４】(i) ランダム偏差法

上に述べてきた凸包構成アルゴリズムでは、同一平面に４点以上がのる場合や同一直線上に３点以上がのる場合には、分岐の判定が微妙になる場合が多数発生する。例えば、上記凸包構成アルゴリズムでは、２頂点を結ぶ直線に対して、ある頂点が上下どちらに存在するかを判定する関数 ConvexJudge2Dim( )、また、３頂点を結ぶ平面に対して、ある頂点が上下どちらに存在するかを判定する関数ConvexJudge3Dim( )を至るところで使用しているが、同一平面に４点以上がのる場合や同一直線上に３点以上がのる場合には判定が微妙になってアルゴリズムが無限ループに陥る場合がある。

【００４５】この状況を避けるため、各頂点の座標値v(0),v(1),v(2)にランダムな微小偏差random()をのせる。具体的には、図７で定義したクラスVertexの３次元ベクトル要素ｄｖを以下のように設定する。各頂点に対して、

ｄｖ（０）＝ｖ（０）；

ｄｖ（１）＝ｖ（１）　＋random( );

ｄｖ（２）＝ｖ（２）　＋random( );

即ち、Ｘ軸成分はそのままにし、Ｙ，Ｚ座標の値に偏差をのせる。Ｘ軸成分を不変にする理由は、Ｘ軸に対する頂点のソーティングがｖ（０）に基づいて行われるためである。各頂点に対してランダム偏差をのせるプロセスは、具体的には、図９の１次元凸包構成 MakeConvex1Dim(com)の過程で行われる。

【００４６】(j) バックアッププロセス

上記凸包構成アルゴリズムのWrapping processでは、Upper Bridgeの頂点utop,vtopから出発してこれらの頂点に戻って来ずに、無限ループに陥る場合がある。この原因は、(i)で述べたものと同じであるが各頂点にのせたランダム偏差が万能である保証はない。Wrapping processにおける無限ループは、Wrappingの途中段階で小ループを構成し、utop,vtop以外のある頂点を繰り返すパターンを取る。utop,vtop以外の頂点を２度以上選択したか否かはハッシングで判定することができる。従って、小ループが発生した時点でWrapping processのメインループを抜け出し、図２２で示すバックアッププロセスに以降する。

【００４７】本プロセスでは、図２２で示すように、conLeft->vcnt, conRight->vcntを比較し、頂点数の少ない方の凸包を解除して、もう一方の凸包に対して解除した凸包の頂点を一点、一点付け加えていくことを行う。頂点を付け加える過程は、頂点を一点からなる凸包と見做して、マージプロセスを繰り返す。頂点を併合していく順序は、conRightの凸包を解除した場合にはconRightの頂点を小さい順に併合していき、conLeftの凸包を解除した場合には大きい順に併合していく。本プロセスでは、一方の頂点が不動であるため無限ループに陥ることは殆どなく、確実な凸包併合が可能となる。一度、併合凸包を構成した後は、以前のマージプロセスに復帰し、以後、Wrapping processで無限ループが再び発生しない限り、通常のマージプロセスが続けられる。従って、バックアッププロセスに伴う計算負荷の増大は、最小限に抑えられる。

【００４８】（Ｂ）干渉チェック前処理（近接点線形リスト）

凸多面体同士の干渉チェックにおいて、最近点を連続的に探索していくための前処理である。この前処理を施すことにより、連続探索が極めて高速化される。通常、この前処理は、図１５に示す通り、凸包構成をした後に実行される。近接点線形リストとは、ＣＧモデルを構成する各頂点に対し、その頂点とエッジを介して繋がっている他の頂点とを線形リストで繋ぐデータ構造のことであり、branch方向にその頂点の近傍頂点、next方向に凸包を構成する頂点が線形にリンクされる。

【００４９】(a) 近接点線形リストの例

図２３は、直方体の近接点線形リストの例を示している。図示の通り、近接点線形リストは、頂点を順次繋いだnext方向と、各頂点にポリゴン辺を介して連結する近接点を配列するbranch方向の２方向を持つ。branch方向の先頭には、その頂点自身が入る。

【００５０】(b) 近接点線形リスト構成アルゴリズム

図２４は対象物体の凸要素毎のポリゴンデータ説明図、図２５は近接点線形リスト構成アルゴリズムの説明図である。対象物体が三角形ポリゴンの集合で構成されているものとすると、近接点線形リストは以下のフローに従って構成される。

1) 凸多面体のポリゴンデータを用意する(図２４参照)。

2) 各々のポリゴンデータ(ｐｉ，ｑｉ，ｒｉ)に対し、各頂点をリストＬに繋げていく。next方向を頂点に対する線形リスト方向、branch方向を各頂点の近接点方向とす

る。リストＬの先頭要素は第１番目のポリゴンの第１頂点であり、作成途中では図２５(a)に示すデータ構造となっている。具体的には以下のようにリストを作成する。

2-1) リストＬの先頭要素topを割り付ける。すなわち、topとして最初のポリゴンの最初の頂点を割り付ける。

2-2) ポリゴン(ｐｉ，ｑｉ，ｒｉ)に対し、以下の割付けを繰り返す。

1) if(ｐｉが全てのnext方向頂点Ｌｊに対し、ｐｉ≠Ｌｊ)ならば、新しい要素ｐｉをnext方向の端にＬｊ+1として割り付ける(図２５(b)参照)。ついで、頂点Ｌｊ+1のbranch方向に対し、自分自身ｐｉ及びｑｉ，ｒｉを割り付ける(図２５(c)参照)。

【００５１】2) if(ｐｉがあるnext方向頂点Ｌｊに対し、Ｐｉ＝Ｌｊ)ならば、頂点Ｌｊのbranch方向に対し、ｑｉ，ｒｉを割り付ける。もし、branch方向に既にｑｉ，ｒｉと同一頂点があれば割り付けることはしない(図２５(d)参照)。

3) 頂点ｑｉに対して上記1)、2)の処理を実行する。ただし、頂点ｑｉに対するｑｉ以外のbranch候補はｐｉ，ｒｉである。

4) 頂点ｒｉに対して上記1)、2)の処理を実行する。ただし、頂点ｒｉに対するｒｉ以外のbranch候補はｐｉ，ｑｉである。

2-3) 全てのポリゴン集合∪ｉ(ｐｉ，ｑｉ，ｒｉ)に対し、2-2)の処理を繰り返す。

【００５２】尚、近傍点線形リストを構成するためのクラス構成は、図７とは別の形式を取るとする。図７のクラス構成は、凸包構成以外に使うことはない。凸包構成の過程において対象を覆う各ポリゴンは自動的に三角形分割されるため、頂点同士を結ぶエッジは既に構成され、図７で示したデータ構造を有しているとする。近接点線形リストを作成するアルゴリズムは、再帰的な手法である。凸包　conの最大エッジcon->edgeから出発し、con->edge->oriq，con->edge->sym->oriq，con->edge->onext->sym->oriqの近接点線形リストを構成し、続いて、エッジcon->edge->sym，con->edge->lnex->sym，con->edge->onextから出発して再帰的に各頂点をサーチしていく。同一頂点に対する２度以上の検索を避けるため、各頂点を検索する毎にハッシングを行う。

【００５３】(c) ダンプファイルのフォーマット

ダンプファイルのフォーマットは、図２６に示すような形式とする。このファイルフォーマットにおいて、高速ローデングの必須項目は以下の通りである。

1) 凸包を構成する総頂点数(count)の項目があること。

2) 各々の頂点に番号付けがされること。

3) 各々の頂点に対し、その頂点の近接点個数(bcount)を格納する項目があること。

4) 各々の頂点座標を格納する項目があること。

【００５４】(d) ダンプファイルからのローディング法

1) 凸包の総頂点数countを読み、各頂点を収めるcount次元の配列を作る。

2) 配列に凸包の各頂点を収めていくと共に、各頂点間をnext方向に結び線形リストを構成する。例えば、頂点を収める配列をＶ〔N_1〕とする時、Ｖ〔０〕，Ｖ〔１〕，．．．．，を順にnext方向に結び、線形リストを構成する。

3) 凸包の各頂点に対し、Branch方向の近接点番号(配列内の位置を指定)を頼りに、Branch方向に近接点のリストを張る。例えば、凸包の全頂点を収める配列をＶ〔N_1〕とする時、頂点Ｖ〔０〕のBranchの番号が(0,9,1,7,8)ならば、Ｖ〔０〕のBranch方向にＶ〔０〕，Ｖ〔９〕，Ｖ〔１〕，Ｖ〔７〕，Ｖ〔８〕を線形リストに繋ぐ。

上記のローディング法では、同じファイルに対して計二回（ 1),2)が１回目、3)が二回目）の読み込みを行う。各頂点のBranch数は殆どの場合において数１０個以下で抑えられるため、合計のローディング時間はＯ（Ｎ）（凸包の総頂点数）である。凸包に対する図２６の形式のファイルをＨＬファイルと呼ぶ。

【００５５】(C) 非凸多面体の第１の干渉チェック

以上の前処理により非凸多面体の凸包及び該凸包の近接点線形リストが作成されれば、特願平6-209008号で提案しているGilbert法による干渉チェックを適用して凸包同士の干渉チェックを行い、凸包同士が干渉する場合には凸包を解除して非凸多面体を構成する凸多面体間の干渉チェックを後述するBubble collision法に従って行う。尚、凸包同士の干渉チェックに際しては、それ迄に求まっている最接近点に近接する頂点を近接点線形リストより抽出し、これら頂点にGilbert法による干渉チェック法を適用して所定時間後の最接近点を探索する。このようにすることにより、干渉チェックに要する時間は凸多面体を表現するポリゴン数にほとんど依存しない一定数以下に抑えることができる。

【００５６】(D) Bubble Collision 法による干渉チェックの前処理

凸包同士が干渉し始めたら、バブルコライジョン法(Bubble Collision 法)を適用して非凸多面体間の干渉チェックを行う。Bubble Collision 法では、非凸多面体を構成する各ポリゴンをポリゴン上に中心を持つ複数の球の組み合わせで覆い、それらの球を包含する球を階層的に構成する。かかる球間の干渉チェックを階層的に行い、最も接近したポリゴンの組み合わせを高速に探索し、該ポリゴン同士の干渉チェックを行うことで非凸多面体間の干渉チェックを行う。

【００５７】かかるBubble Collision 法を適用するには、非凸多面体を構成する各ポリゴンをポリゴン上に中心を持つ複数の球の組み合わせで覆い、それらの球を包含する球を階層的に構成する前処理が必要になる。階層的に配置された球集合のツリーをSphere Treeと呼ぶ。S

phere Treeの構成は、分割統治法で行われ、ＮポリゴンからなるＣＧモデルに対して、Ｏ（Ｎ・log Ｎ）の時間で処理が終わる。本処理とQuinlan法におけるSphere Treeの構成法の違いは、Quinlan法ではスキャン変換(scan conversion)に類似した方法であるのに対し、本手法では、対象を覆うポリゴン数を考慮し、スケールＳの値をパラメータとした効率的な構成を行う。

【００５８】(a) Leaf Spheresの構成法

干渉チェックの対象となる非凸多面体は三角形ポリゴンの集合で覆われているとする。この時、三角形ポリゴン上に中心を持ち、対象を覆う球集合の最も底辺に位置する基本球をLeaf Sphereと呼ぶ。図２７はLeaf Spheresの例を示している。Leaf Spheresの構成法は以下の通りである。

1) 初期設定

入力データは各三角形ポリゴンの頂点座標、スケールパラメータＳとする。スケールパラメータＳの値の設定法は後述する。図２８に示すように、三角形ポリゴンの頂点座標を基に辺の長さ $d_0=|p_1-p_0|$, $d_1=|p_2-p_1|$, $d_2=|p_0-p_2|$を計算した時に、$d_0 \geq d_1 \geq d_2$になっているとする。もし、これを満たさなければ、頂点番号を入れ換える。辺p_0p_1上における単位ベクトル $e_0=(p_1-p_0)/d_0$を設定する。三角形ポリゴンの高さをｈを $h=\sqrt{\{d_2^2-((p_1-p_0)\cdot(p_0-p_2)/d_0)^2\}}$ で計算する。

【００５９】2) メイン処理

2-1) 辺p_0p_1に沿ってLeaf Spheresを作成する。$N_0=[d_0/S]$を計算し、もし$N_0=0$ならば、半径 $r_0=d_0/2$とする。$N_0 \neq 0$ならば $r_0=d_0/(2\cdot N_0)$とする。従って、この時には辺P_0P_1に沿って半径r_0の球がN_0個並ぶ。

2-2) もし、($r_0>h$)ならば、辺p_0p_1上のLeaf Spheresで三角形ポリゴン全体が覆われているとし(図２９参照)、処理を終了する。

2-3) もし、($r_0 \leqq h$)ならば、単位ベクトル $e_1=(p_2-p_1)/d_1$, $e_2=(p_0-p_2)/d_2$ および $N_1=[d_1/S]$, $N_2=[d_2/S]$を計算し、辺p_1p_2、辺p_2p_0に沿って、辺p_0p_1と同様にLeaf Spheresを作成する。

【００６０】2-4) ついで、三角形ポリゴンの内部をLeaf Spheresで覆う。図３０に示すように三角形ポリゴンの底辺を直径 $diam_0=2\cdot r_0$だけ底上げし、p_0p_1に沿って作成したのと同様にして底上げした底辺上に Leaf Spheresの球中心を配置する。球の半径ｒは、今回計算した半径r_iと一つ前の半径r_{i-1}と大小比較し、大きい方を選択する。即ち、$r=\max(r_{i-1},r_i)$により更新していく。これにより、Leaf Spheresを積み上げていった時の上下間の Leaf Spheres同士の隙間を小さくすることができる。

2-5) 上記2-3)、 2-4)の処理を底上げした三角形の高さ $h_i=h_{i-1}-diam_{i-1}$が$h_i<diam_{i-1}/2$を満たすまで繰り返す。

【００６１】(b) 分割統治法によるSphere Treeの構成法

Sphere Treeとは、Leaf Spheresを要素として、それらを包含する球を階層的に配置したツリー状データ構造のことである。ツリーの分岐の度数としては、二分木をとる。この理由は、任意の非凸多面体に対しては、二分木による構成が最もバランスする（各階層の枝下に含まれるLeaf Sphereの個数がバランスする）可能性が高いためである。二分木以外に例えば八分木を採用したとすると、元々、対象の形状が８象限に対してバランスしていない限り各階層における八分木の各枝下にLeaf Sphereがほぼ等しくちりばめられる可能性は低い。

【００６２】図３１はSphere Tree構成のためのデータ構造、図３２はそのフローを示している。Sphere Treeの基となる三角形ポリゴン上に中心を持つ球をLeafで表し、Leafの集合をLeavesで表す。Sphere Treeの各枝に配置される球要素はSphereで表す。Leaf、Sphereとも球の中心をcenter,半径をrに格納する。Leafのメンバplane* planeは、そのLeaf Sphereの中心がのる三角形ポリゴンをポイントする。Leavesのメンバmax,minにはLeaf* leavesに連なるLeaf要素 iのcenterを(x_i,y_i,z_i)とする時、以下の値が入る。

max=($MAX_i\{x_i\}$, $MAX_i\{y_i\}$, $MAX_i\{z_i\}$)

min=($MIN_i\{x_i\}$, $MIN_i\{y_i\}$, $MIN_i\{z_i\}$)

また、leafcntには、Leaf* leavesに連なるLeaf要素数が入り、avcenterにはLeaf要素の中心座標の和を要素数で割った重心が入る。Sphereのメンバleft, rightは二分木を表し、Leavesはその階層以下に含まれるLeaf要素の集合をポイントする。

【００６３】Sphere Tree構成アルゴリズムの概要は以下の通りである。

1) Planesのnext方向にリニアにつながれたポリゴン情報を入力として、各ポリゴン毎にLeaf要素の集合を作り、それら全体をポリゴン集合全体にわたりリニアにリンクする（図３２、ステップ２００１）。

2) Leafのリニア集合leaves->leavesを入力として、分割統治法により、Sphere Treeを構成する。

2-1) Leaf要素数が０または１の場合は、ステップ２００２、２００３の処理を実行する。

2-2) Leaf要素数が２以上の場合は、二分木のメモリを確保する(ステップ２００４)。

【００６４】2-3) Leaf要素集合leaves->leavesを分割するために、leaves->leavesに対するmax,minを計算し(ステップ２００５)、Leaf要素集合がＸ軸、Ｙ軸、Ｚ軸のどの軸方向に最も伸展しているかを判定し(Leaf要素数が最大の軸方向を求める)、伸展方向の中心値を算出する(ステップ２００６)。

2-4) Leaf要素集合を、2-3) で求めた軸方向に対して中心値を境に分割する。(ステップ２００７)。

2-5) 分割した各々の Leaf要素集合に対して、再帰的に分割統治法を行う(ステップ２００８)。

2-6) 再帰的に求めた left, rightの重心avcenterと要素数leafcntを基に、親のSphereのavcenterとleafcntを求める(ステップ２００９)。

2-7) left, rightをマージ(併合)し、親のSphereを構成する(ステップ２０１０)。

【００６５】親のSphereを構成するに当たり、その半径と中心の求め方として二通り用意する。第一の方法は、子Sphereのleft,rightを包含する最小の球を構成する方法、第二の方法は、2-6)で求めたavcenterを中心として、その階層以下に含まれるLeaf要素集合を包含する最小の球を構成する方法である。親Sphereとしては、上記二つの球を求め、半径の小さい方を採用する。図３３は上記アルゴリズムに基づき、直方体に対して求めたSphere Treeの例を示している。

【００６６】(c) スケールパラメータＳ

対象をいくつのLeaf Spheresで覆うかは、対象を覆うポリゴン数Ｎに大きく依存する。Ｎが充分大きい場合には、一つのポリゴンを細かい球で覆う必要はなくなる。一方、Ｎが小さい場合には、適度にLeaf Spheresの数を増やし、球による被覆の近似を上げる必要がある。境となるポリゴン数は丁度１００前後である。スケールパラメータＳは以下のアルゴリズムに従って計算する。

1) 凸包を構成する頂点集合に対して、

$max=(MAX_i \{x_i\}, MAX_i \{y_i\}, MAX_i \{z_i\})$

$min=(MIN_i \{x_i\}, MIN_i \{y_i\}, MIN_i \{z_i\})$

を計算し、凸包を包絡する球の半径Ｒ＝｜max－min｜／２を求める。

【００６７】2) 対象のポリゴン数があるしきい値 PLANELIMITより小さい時には、あるスケール定数 SPHERESCALEを使って、

Ｓ＝Ｒ／SPHERESCALE

とする。他の場合には、

Ｓ＝Ｒ

とする。 PLANELIMIT, SPHERESCALEの典型的な値は、PLANELIMIT＝１００, SPHERESCALE＝８である。

【００６８】（Ｄ）非凸多面体干渉チェックにおけるBubble Collision法

非凸多面体の干渉チェックを高速に行うために、各々の非凸多面体をSphere Treeで包絡し、そのツリーをDepth-firstの探索法でチェックしていき、最も接近するSphereのペアを検索する。検索したSphereの中心が載っているポリゴンの組み合わせに対してGilbert法を適用をすることにより、二つの非凸多面体の最近点を算出する。本手法とQuinlan法との違いは、Sphere間の距離の計算法にある。本手法では、Sphere間の距離を求めるに際し、そのSphereが Sphere Treeのどの階層にあるかによって、Sphere間の距離の定義を変更する。これにより、Bubble Collision法で問題となっている、最終的にチェックしなければならないポリゴンペアの数を抑えることができる。

【００６９】(a) Depth-firstによる最近点探索法

二つの非凸多面体の最近点を以下のDepth-firstアルゴリズムに従って求める。尚、図３４、図35は Bubble Collision法における Bubble Collision Checkの基本関数である。

1) 二物体間の距離の初期値cq->distを無限大に取る。ここで、cqは干渉チェックにおける環境クラスの変数を表す。詳細は後述する。

2) 今、着目しているSphere同士の距離を求め、その値が cq->dist以下ならば、カレントの(着目している)Sphereを以下の方針に従って縦方向に解除していく(ステップ２１０１)。

2-1) カレントのSphereが両方ともLeaf Sphereでない時には、半径の大きい方を解除する(ステップ２１０２、２１０５)。

2-2) 解除した結果得られる二つの子Sphereと解除しないSphereとの距離を求め、近い方を先に探索する(ステップ２１０３、２１０４)

2-3) カレントのSphereの一方が Leaf Sphereである時には、Leaf Sphereでない方を解除し、2-2)と同様に探索する(ステップ２１０６、２１０７)。

【００７０】3) カレントのSphereが両方ともLeaf Sphereである時には(ステップ２１０８)、それらのLeaf Sphereがのっている三角形ポリゴンペアに対してGilbert法を適用し、三角形ポリゴンペア間の距離distを求める（ステップ２１１０)。

4) 求めたdistの値がcq->distより小さければ、cq->distの値を更新し、上記の処理を再帰的に繰り返す(ステップ２１１１)。

上記アルゴリズムでは、3) の過程において、同一の三角形ポリゴンペアによる干渉チェックが多数回発生する。そのため、ポリゴンペアに対してハッシングを行い、不必要な干渉チェックを行わないようにしている、（ステップ２１０９)。

【００７１】(b) Sphere間の距離

Bubble Collision法の効率を上げるには、cq->distの値を出来るだけ速やかに小さくし、Depth-firstにおける探索回数を減らすことが必要である。探索回数を減らすために、Sphere間の距離を以下のように定義する。

1) Intermediate Sphere(中間球)の間の距離は、中心間の距離から両方の半径を引いた値(図３６の❶)とする。

2) InrermediateSphereとLeaf Sphere間の距離は、中心間の距離からIntermediate Sphereの半径を引いた値(図３６の❷,❸)とする。

3) Leaf Sphere間の距離は、中心間の距離(図３６の❹)とする。

この定義により、Sphere Treeの他の枝における探索の深度を浅く留めることができ、また、図35のステップ２１１０において同一ポリゴンペアが発生することもなくなる。

【００７２】但し、この定義には、一つだけ難点がある。それは、二物体が複数の点で接触する多点接触の場合である。この場合には、cq->distの値が、ある接触点のために０に近い値を取ってしまうため、他の接触点の探索においてその点を含むSphereを見逃してしまうことがある。これを避けるため、Sphere間の距離に、以下の定義を付け加える。

4) もし、cq->distの値がある微少量ＥＰＳより小さい場合には、Sphere間の距離を1) の定義とする。微少量ＥＰＳの典型的な値は、$1/10^4$である。

【００７３】(c) 計算負荷

Bubble Collision法の計算負荷は、最も効率良い場合に$O(\log N_0 + \log N_1)$（N_0，N_1はそれぞれ各対象を被覆しているLeaf Sphereの総数）である。被覆の度合い、即ち、Leaf Sphereの個数は多すぎても少なすぎてもいけない。少なすぎるとcq->distの値がなかなか小さくならず、探索回数が増えてしまう。また、多すぎる場合には、探索の深度自身が深くなる。効率を上げるための現実的な方法としては、対象を表現するポリゴン数が大きい時には、１ポリゴン当たり１Leaf Sphereで覆い、ポリゴン数が少なくなるに従い、Leaf Sphere数を増やすようにする。具体的には、前述のスケールパラメータＳに従って被覆の度合いを調整する。

【００７４】計算負荷を下げる他の方法では、相対誤差パラメータcq->alphaの導入である。図35のステップ２１１１に示すようにcq->alphaを導入すると、cq->distの更新値は、求めたばかりのポリゴン間の距離distより小さく設定され、他の枝における探索の条件が厳しくなる。cq->alphaの値は０から１の実数を取り、求めた距離distに対して誤差を導入する効果がある。最近点間距離の真値をｄとすると、$dist \geq d \geq \cdot dist \cdot (1 - cq->alpha)$ の関係が成り立つ。相対誤差パラメータは、対象同士が大きく離れている場合には効果が大きく、近づくにつれて効果が薄れ、接触している場合には、cq->alpha＝０の場合と全く変わらなくなる。

【００７５】（Ｅ）凸多面体干渉チェックにおける凸包間連続型Gilbert法と非凸多面体用Bubble Collision法の融合。

非凸多面体間の干渉チェックを出来るだけ無駄なく行うために、非凸多面体間が充分離れている場合には非凸多面体を包含する凸包を構成し、凸包間に連続型Gilbert法による干渉チェックを適用する。凸包同士が干渉を始めたならば、Bubble Collision法に自動的にスイッチする。スイッチングの指標は、凸包間の距離とする。

【００７６】(a) 干渉チェック用データ構造

図３７，図３８，図３９は本干渉チェックアルゴリズムを表現するために必要となる干渉チェックデータ構造の中枢部分を示している。CollisionQueは干渉チェック用の環境を定義するクラス、ColQue干渉チェックを行う対象を表現するクラス、Primsは対象を構成する頂点を表現するクラス、ConvexHullは図３２に示すＨＬファイルから構成した凸包を表現するクラス、MetaLeaf，MetaLeaves，MetaSphereは、後ほど説明するメタツリー関係のクラスである。

【００７７】今、問題としている対象を二つの組み分け、それぞれ、ColQueリスト１、ColQueリスト２とする。各々のColQueリストは、クラスColQueのnextポインタを介してリニアにつながれ、それらの先頭要素が環境クラスCollisionQueのnext，next2につながれている。この時、本アルゴリズムは、ColQueリスト１、ColQueリスト２の間で最も近接するポイントを捜し出す。ColQueリスト１内、ColQueリスト２内での干渉チェックは行わない。例えばColQueリスト１としてロボット１の各リンク、ColQueリスト２としてロボット２の各リンクを設定すると、ロボット１とロボット２の間で干渉チェックを行い、ロボット１の各リンク同士、ロボット２の各リンク同士の干渉チェックは無視される。ロボット１のリンク数をＬ１，ロボット２のリンク数をＬ２とすると単純には、Ｌ１×Ｌ２回の干渉チェックを行わなければならないが、この過程を高速化する方法がメタツリーである。これについては後述する

【００７８】(b)Gilbert法の概要

凸多面体同士の干渉チェックを頂点数に対してリニアの計算量で実行するアルゴリズムがGilbert法である。図４０は、Gilbert法による最近点算出法の基本的考えを示している。尚、詳細は特願平6-209008号公報(最接近点探索方法及びその前処理方法)を参照されたい。Gilbert法では、まず、対象の重心同士を結んだベクトルv_0を作る。$-v_0$（およびv_0）と各頂点との内積に対して、最大値を計算することにより、最近点の候補p_0，q_0を見つけ出す(図４０(a))。v_0をp_0，q_0を結んだベクトルv_1に更新し、同様の内積計算により新たな候補p_1，q_1を見つけ出す(図４０(b))。一般にv_iを頂点集合$\{p_k\}$が張るポリゴンと$\{q_k\}$が張るポリゴン間の最近ベクトルとすると、上記処理は数回の繰り返しにより収束し、最終的に各対象上における最近点および最近ベクトルv_{final}が得られる(図４０(c))。$\{p_k\}$,$\{q_k\}$が張る図形は何れも三角形、エッジまたは１頂点であるため、この間の距離は簡単に求められる。上記のように、Gilbert法の計算負荷は、対象を構成する各頂点との内積計算により発生する。従って、その計算オーダは$O(n_1 + n_2)$（n_1，n_2：各対象の頂点数）である。

【００７９】(c) 連続型Gilbert法

最近点を連続的に求めていく場合には、前ステップで求めた最近点の情報を使うことにより一層、効率的に次の最近点を計算することができる。図４１は本アルゴリズムの概念図を示している。上記(b)で説明したようにGilbert法では、毎回、全点にわたる内積計算を繰り返す。最近点を連続的に追跡していく場合、次ステップにおける最近点の位置は前ステップにおける最近点の近傍に存

在するはずである。従って、Gilbert法の内積計算は、最近点の近傍のみで充分である。具体的にはv_{final}(図４０）を構成する頂点集合$\{p_k\},\{q_k\}$の各々に対して、ポインタ変数branch方向をサーチして近傍頂点の和集合をつくり、これに対して内積を取る。近傍頂点の和集合を構成するに際しては、ハッシングを使い同一頂点の重複を避けるようにする。

【００８０】連続型Gilbert法の計算負荷は、近傍頂点の和集合の要素数に比例する。この値は、厳密に言えば、対象を表現するポリゴン数およびそのパッチングの仕方に依存する。例えば円錐や円柱の底面を非常に細かく花びら状にパッチングした場合には、中心付近における近傍頂点和集合の要素数は、他の領域に比べ格段に大きくなる。しかしながら、このような例外を除き、他の多くの場合において、要素数は充分小さい値（具体的には２０～３０以下）に抑えられる。従って、連続型Gilbert法の計算負荷は、対象に依らず一定値以下ということができる。16Mflopsの計算機マシーンを使ったラフな評価では、殆どの対象に対して0.5m s／cycleの性能がでている。

【００８１】(d) 融合アルゴリズム

図４２は連続型Gilbert法とBubble Collision法の融合アルゴリズムのフローを示している。凸包の外側では、連続型Gilbert法による干渉チェックを行い(ステップ３００１)、干渉しなければステップ３００２で「ｎｏ」となり、その結果を最近点とする(ステップ３００３)。凸包同士が干渉チェックしだしたならば、ステップ３００２で「yes」となり、Bubble Collision法に移行し(ステップ３００４）、その結果を最近点とする(ステップ３００５)。再び、対象間の距離が離れ、連続型Gilbert法の算出する距離が０より大きくなったならば、ステップ２００２で「ｎｏ」となり、Bubble Collision法をやめ、連続型Gilbert法の結果を最近点とする。上記アルゴリズムにおける難点は、対象同士が干渉しだし、一方が他方を飲み込むような状況においても連続型Gilbert法が距離０を算出するか否かである。これに関しては、３次元のGilbert法を４次元に拡張することにより解決する。

【００８２】(e) ４次元Gilbert法

凸包間連続探索法から非凸多面体用Bubble Collision法へスイッチングする場合、またはその逆において、スイッチングの指標を凸包間連続型Gilbert法により算出した距離とする。凸包間が干渉しだしたならば、距離を算出するアルゴリズムを３次元連続型Gilbert法から（３＋Extra－Dim.）次元、すなわち、４次元連続型Gilbert法にスイッチングする。これにより、一方の凸包が他方の凸包に吸収された場合においても近接ベクトルが矛盾なく計算される。図４３は、上記状況を(２＋Extra－Dim.）次元の場合に示したものである。図４３の❶)に示すように、二つの凸包が干渉していない場合には、凸包間の距離はＸ－Ｙ平面上の２次元的距離ｄで定義される。ｄの値は、２次元連続型Gilbert法で計算される。ｄの値がある微少量より小さくなったならば、二つの凸包は干渉を始めたとし、一方の凸包をExtra Dimensionの方向にεだけ嵩上げする(図４３の❷、❸)。この状態における近接ベクトルは、3次元連続型Gilbert法を使って計算される。この時、近接ベクトルはExtra Dimensionの方向に向いており、Ｘ－Ｙ平面上の成分は０である。従って、二凸包間の距離ｄは常に０であり、二凸包は干渉状態にあると判断される。近接ベクトルのＸ－Ｙ平面成分が非零になったならば、二凸包の干渉状態は解除されたと判断し、干渉チェックは（２＋Extra－Dim.）次元、すなわち、３次元の連続型Gilbert法から２次元連続型Gilbert法へ移行する。Extra Dimension方向の嵩上げεは０以外の任意の値でよい。

【００８３】(f) メタツリーによる高速化

非凸多面体の集合において、最も近接する非凸多面体のペアを高速に算出するために、非凸多面体を包絡する球を構成し、この球を包含する包絡球を階層的に並べたメタツリーを構成する。非凸多面体間を総当たり的にチェックする代わりに、メタツリーに基づいて包絡球間をdepth-firstの探索法でチェックしていき、最も接近する球のペアを検索する。検索した球の中心が載る凸包間に連続型Gilbert法を適用することにより、最近点を効率よく算出する。図４４は、ロボットアームＲＡ１，ＲＡ２に対してメタツリーを構成し、それを二本並べて干渉チェックを行っているところを示している。メタツリーを構成するに際し、Leaf Sphereに相当する球は以下で定義される。

【００８４】干渉チェックを行う各非凸多面体に対し、ベクトルmax,minを

$max=(MAX_i\{X_i\},MAX_i\{y_i\},MAX_i\{z_i\})$

$min=(MIN_i\{X_i\},MIN_i\{y_i\},MIN_i\{z_i\})$

(iは非凸多面体の各頂点を走る)

で定義する。この時、球の中心center及び半径ｒは以下で与えられる。

$center=(max+min)/2$

$r=|max-min|/2$

メタツリーの構成法は、Bubble Collision法における、Sphere Tree構成法と同様に行う。メタツリーを構成した後、Bubble Collision法と同様に、Meta Sphere Treeに対してDepth-Firstの探索を行い、干渉の可能性のある非凸多面体のペアを検出する。この時、Meta Sphere間の距離は、両Meta Sphere間の中心間の距離から両半径を差し引いた値とする。

【００８５】(g) メタツリー構成を含めた連続干渉チェック法

任意の非凸多面体間の最近点を連続的に追跡していく場合において、メタツリー構成、Gilbert法、Bubble Collision法の全てを数サイクルに一回のみ行い、他のサイ

クルでは、前回のサイクルで使用した近接ポリゴンペアに再度Gilbert法を適用することにより最短距離を求め、計算負荷を大幅に軽減する。図４５は、本アルゴリズムのフローを示している。まず、サイクルをカウントして現サイクルとし、該現サイクルがサイクル定数CYCLEで割り切れるか否かを判定し、割り切れるならばステップ３１０２～ステップ３１０６の処理を行い、割り切れなければステップ３１０５～３１０６の処理を行う。例えば、CYCLE＝１００ならば、１００サイクル毎にステップ３１０２～ステップ３１０６の処理を行い、それ以外はステップ３１０５～３１０６の処理を行うことになる。

【００８６】すなわち、割り切れなければステップ３１０２において、図４４で説明したメタツリーを構成し、Meta Sphere Treeに対してDepth-Firstの探索を行い、干渉の可能性のある非凸多面体のペアを検出する。この処理により、Gilbert法を適用する凸包のペアが確定し、ステップ３１０３において、該確定した凸包のペアに対してGilbert法による干渉チェック処理を実行する。凸包ペアが前回のペアと同一であれば、ステップ３１０３で行うGilbert法は連続型となる。それ以外は非連続型である。凸包間が干渉していなければ、Gilbert法の結果を近接ベクトルとする（ステップ３１０６）。干渉していれば、ステップ３１０４のBubbleCollision法に移行し、その結果を近接ベクトルとすると共に（ステップ３１０６）、Bubble Collision法の過程で拾いだされる近接ポリゴンペアをステップ３１０５において保存する。現サイクルがCYCLEの倍数でない場合には、ステップ３１０５で保存したポリゴンペアにGilbert法を適用し、最も近接するペアを拾いだし、近接ベクトルを算出する。

【００８７】(h) サイクルタイム自動調整型連続干渉チェック法

図４５の連続干渉チェック法では、CYCLEの値が固定的であった。本アルゴリズムでは、図４５のステップ３１０２～３１０４で費やされる時間を計算機のクロックから直接測定し、その値を基にCYCLEを決定する。図４６はサイクルタイム自動調整型連続干渉チェック処理の流れ図であり、図４５のフローと異なる点は、ステップ３１０２の処理開始時刻t_1を計時するステップ３２０１と、ステップ３１０４の処理終了時刻t_2を計時するステップ３２０１を付け加え、ステップ３１０１において、CYCLEの値を次式

CYCLE＝［A／（t2－t1）］

により動的に決定している点である。ここで、Aは計算機のＣＰＵパワーに依存する定数である。これにより、t2－t1の値が大きい時、すなわち、メタツリー構成からBubble Collision法に至るまでの処理時間が長い時には、CYCLEの値は小さくなり、ステップ３１０２～３１０４の処理を行う頻度は増す。また、逆にt2-t1の値が小さい時には、その頻度は減少し、より計算負荷の小さいステップ３２０５の処理を行う頻度が増す。

【００８８】（Ｅ）連続型Bubble Collision法

(a) リアルタイム干渉チェック処理の全体フロー

図４７は本発明のリアルタイム干渉チェック処理の全体フローである。シミュレーションが開始されると（ステップ４１００）、干渉チェック前処理を行う（ステップ４２００）。凸包間の連続型干渉チェックのための前処理では、凸包ファイル（ＨＬファイル）があるかないかを判定する（ステップ４２１１）。初回は凸包ファイルがないので、呼び込むＣＧモデルに対して凸包を構成し（ステップ４２１３）、凸包ファイル（ＨＬファイル）を出力する（ステップ４２１４）。次からは凸包ファイルがあれば、それを読み込む（ステップ４２１２）。Bubble Collision法の前処理（ステップ４２２０）では、非凸多面体の三角形ポリゴン上にLeaf Sphereを作成し（ステップ４２２１）、それらを階層球で包絡し、包絡球からなる２分木ツリーを構成する（ステップ４２２２）。

【００８９】ついで、移動指令値作成のステップ４３００において、マウス３０２等のマンマシーンインタフェース機器（ＭＭＩ機器）より入力された三次元入力データに基づいてインタラクティブに物体の移動指令値を作成する（ステップ４３１０）。または、プログラミングされた軌道に沿って移動指令値を作成する（ステップ４３２０）。しかる後、移動指令値に従って、図４８に示すように物体４０４（ここでは直方体Ｂ）を物体４０５（ここでは非凸多面体Ａ）に対して移動して（ステップ４４００）、ＣＧ内の環境を更新する。　ここで、マウスを用いた三次元データ入力方式について簡単に説明する。マウスを用いた三次元入力方式のシステム構成図を図４９に示す。このシステムでは、オペレータ３００は、計算機３０１に付属の三ボタンマウス３０２とキーボード３０３上のファンクションキー３０４を使用し、特別な装置を必要としない。計算機３０１のディスクプレイ画面には、図５０に示すＣＧモデルの画像が表示されているとする。図５０では、カーソル４０５の位置に応じて、三次元座標計算部４０６は速度指令値を作成し、それを積分して物体の三次元座標を計算する。マウス３０２のボタン１，２，３（４１１～４１３）をｘ，ｙ，ｚの各々の方向に対応付けてあり、ボタンがＯＮの場合にはカーソル位置に応じて速度Ｖｘ，Ｖｙ，Ｖｚを作成する。カーソル位置が画面中心線Ｘcentでは速度は零で、カーソル変位量Ｖが小さいならば、速度は小さく、カーソル変位量Ｖが大きいならば、速度は大きくなる。

【００９０】更新したＣＧ内の環境について、干渉チェックを行う（ステップ４５００）。物体が離れている場合は、凸包間の連続型干渉チェックを高速に行い（ステップ４５１０）、最近点、干渉点、距離を求める（ステ

ップ４５１１）。凸包間の干渉判定を行い（ステップ４５１２）、凸包間が干渉しない間は凸包間の連続型干渉チェックが続く。凸包間が干渉すると、連続型Bubble Collision法を行い（ステップ４５２０）、凸包が解除された非凸物体に関して最近点、干渉点、距離を求める（ステップ４５２１）。ファンクションキー等により、シュミレーション終了命令が発生すると（ステップ４６００）、シュミレーションを終了する（ステップ４７００）。終了命令が発生しないと移動指令値作成以降の処理を繰り返す。以上がリアルタイム干渉チェックシステムの全体に渡る説明であり、マウスを用いてインタラクティブに物体を移動しながら、逐次、実時間での干渉チェックを行っていくシステムに有効となるものである。または、プログラミングされた軌道を移動しながら、実時間で物体間の干渉チェックを行うシステムにおいても有効となる。以下、ステップ４５２０の連続型Bubble Collision法について詳細説明を行う。

【００９１】(b) Bubble Collision法の前処理

前処理では、多面体４０４，４０５を構成する三角形ポリゴン上に最小球 Leaf Sphereを配置する（図５１）。図５２は、三角形ポリゴンを Leaf Sphereで覆った例であり、(a)はLeaf Sphereが比較的大きい場合、(b)はLeaf Sphereが小さい場合の例である。（b）の場合には、三角形ポリゴン中に隙間があるので、ほぼ等しい大きさの球で中の隙間を埋める。Leaf Sphereの半径は、既述のスケールパラメータＳによって決定される。すべての三角形ポリゴンに対して、上記の処理を行い、物体をLeaf Sphereで埋める。Leaf Sphereを分割統治法を行い階層球で包絡し、階層包絡球の２分木ツリー構造（図５３）を構成する。図５４は階層包絡球の２分木ツリー構造の３次元イメージである。図５５は物体１０４（直方体Ｂ）の場合の階層包絡球の構成例を階層的に示す。図５６は非凸多面体１０５（物体Ａ）の場合の階層包絡球の構成例を階層的に示す。但し、図５５、図５６では、三角形ポリゴン全体についての図は非常に繁雑になるので、Leaf Sphereを輪郭部のみに載せた例で簡略化して示している。

【００９２】(c) 連続型 Bubble Collision法

連続型Bubble Collision法は、Bubble Collision法の高速化を図ったものである。連続型Bubble Collision法における物体の状態を図５７に、連続型Bubble Collision法における処理フローを図５８に示す。例として非凸多面体Ａと直方体Ｂを考える。連続型Bubble Collision法は、複雑な非凸多面体間で物体が凸包内部に入り、かなり接近している場合の干渉チェックに適用される（図５７参照）。

1) 連続型Bubble Collision法を開始すると（ステップ５１００）、第１回目、または、近接ポリゴンペア更新サイクルであったら（ステップ５２００）、Bubble Collision法を行う（ステップ５３００）。そうでなかったら、前の処理で保存されている近接ポリゴンペアの干渉チェックを行う（ステップ５４００）。

【００９３】2) Bubble Collision法

2-1) 近接球ペア探索（ステップ５３１０）では、前処理で構成した包絡球間の干渉チェックを２分木ツリーに基づいてDepth-firstの探索法でチェックしていき、最も接近する球のペアを探索する。図５９は近接球ペアの探索／近接ポリゴンペアの探索の手順を示す。図５９(a)のように２物体Ａ、Ｂの上位階層の包絡球が干渉すると、その下位階層の包絡球間の干渉チェックを行う。包絡球が干渉した部分のみ、更に、下位階層の球間の干渉チェックを行っていく。すると図５９（g）のように、最も接近する球のペアが探索される。

2-2) 近接ポリゴンペア探索ステップ（ステップ５３２０）では、探索した球の中心が載っているポリゴンのペア（図５９(g)のＰ１１とＰ１２、Ｐ２１とＰ２２）を求める。これらは最近点である可能性のあるポリゴンのペア（近接ポリゴンペア）である。

【００９４】2-3) 近接ポリゴンペアの保存ステップ（ステップ５３３０）はステップ５３２０で得たポリゴンＰ１１とＰ１２、Ｐ２１とＰ２２等の近接ポリゴンペアを保存する。

2-4) 干渉チェックステップ（ステップ５３４０）では、その近接ポリゴンペア間に対してGilbert法を適用することにより、二つの非凸多面体間の干渉点、最近点、距離を算出する。干渉すれば（ステップ５３４１）、干渉点を出力する（ステップ５３４２）。非干渉時には、ポリゴンＰ１１とＰ１２間の干渉チェックから最近点の距離ｄ１を、ポリゴンＰ２１とＰ２２間の干渉チェックから最近点の距離ｄ２、・・を求める（ステップ５３４３）。しかる後、最も短いものを最近点距離とする（最小値判定、ステップ５３４４）。図５７(b)では距離ｄ２が距離ｄ１より小さいのでｄ２を最近点距離の解とする。最近点を出力する（３４５）。

【００９５】3) 近接ポリゴンペアの干渉チェック

3-1) 次のサイクルで図５７(c)のように直方体Ｂを左方向に移動する。しかし、近接ポリゴンペア更新サイクルでないので、保存されている近接ポリゴンペア間の干渉チェックを行う（ステップ５４００）。直方体Ｂが僅かに移動したと仮定すると、その近接ポリゴンペアＰ１１，Ｐ１２；Ｐ２１，Ｐ２２は同じである。そこで、先に保存した近接ポリゴンペア間に対してのみ干渉チェックを行って、最近点、干渉点を求める。近接ポリゴンペアの呼出しステップ５４１０では、先にステップ５３３０で保存した近接ポリゴンペアＰ１１，Ｐ１２；Ｐ２１，Ｐ２２等の近接ポリゴンペアを呼び出す。

3-2) ここで得た近接ポリゴンペアＰ１１,Ｐ１２;Ｐ２１,Ｐ２２等に干渉チェックを行って干渉点、最近点を求める（ステップ５４４０）。図５７(c)では距離ｄ３が距離ｄ２より小さいのでこれを最近点距離の解とす

る。

【００９６】移動距離／姿勢変化が小さいという仮定から、しばらくは、この数組の近接ポリゴンペア間に対してのみ干渉チェックを行う近接ポリゴンペアチェック（ステップ５４００）を繰り返す。保存した数組の近接ポリゴンのみに関して干渉チェックを行うことにより、計算時間の短縮が図られる。しかし、移動距離が大きくなると近接ポリゴンペアが変わる可能性があるので、数サイクル毎にBubble Collision法（ステップ５３００）を行って近接ポリゴンペアを更新する。ＣＧ上での最近点の表示は、移動距離が大きくなると一時、誤った点を表示するが、すぐに次の近接ポリゴン更新サイクルで修正される。更新サイクルが適切ならば、一時、誤った結果が表示されても、瞬時に修正された正しい結果が表示されるので、使用上に問題ない。本実施例では、近接ポリゴン更新サイクルを使用上に問題ないように調整した値を手で入力したものを用いる。

【００９７】(d) 連続型Bubble Collision法の効果

図６０（a）はBubble Collision法のみの場合における干渉チェック計算時間であり、常に計算時間が数十～数百ｍｓかかっている。一方、本発明によれば干渉チェック計算時間は図６０（b）に示すようになる。Bubble Collision法では計算時間が数十～数百ｍｓかかるが、近接ポリゴンペアチェックは数ｍｓであるので、大半はこちらを行うと平均計算時間は点線のように小さくなる。例えば、Bubble Collision法による処理時間を１００ｍｓ、近接ポリゴンチェックの処理時間を１ｍｓ、更新サイクルを１００とすると、平均計算時間は２ｍｓ（＝(100+99･1)/100)程度になる。もちろん、任意の非凸多面体にもこの連続型Bubble Collision法を適用できる。連続型Bubble Collision法によって、任意の非凸多面体に対して物体移動速度に応じて逐次、実時間での干渉チェックが可能になる。一例として、マウス等のＭＭＩにより、インタラクティブな動作時の干渉チェックをリアルタイムに行い、その結果をＣＧ上に表示してユーザに提示する環境の構築への利用等を可能にする。機構設計用ＣＡＤシステムに適用すると、ＣＧ上で実時間で干渉チェックを行いながら部品の組み立てを行うことで、本当に組み立てられるかのチェックができる。また、組み上がった機構部の移動チェックが行える。これにより、設計ミスを発見できるので設計へのフィードバックが早くなることや、試作品を減らす効果が得られる。他にもマニピュレータや自走車等の移動ロボットのパス生成、マルチメディアにおけるアニメーション作成、ゲームソフト等、コンピュータグラフィックスを応用した様々な分野に適用される。

【００９８】（Ｆ）連続型Bubble Collision法の更新サイクル自動決定法

連続型Bubble Collision法では、物体形状と移動速度に応じて近接ポリゴンの更新サイクル（次のBubble Collision法実行までのサイクル）の調整が必要である。

（Ｅ）の実施例では、近接ポリゴン更新サイクルとして、調整した一定値を手で入力したものを使っているが、この更新サイクルが不適切な場合に次の問題が生じる。例えば、更新サイクルを２００毎にするとサイクルが長すぎて、修正が遅くなり、図６１（b）のように間違ったポリゴンペアＰ２１,Ｐ２２間の干渉チェックを行い、それらの最近点距離ｄ２´を出力してしまう。一方、サイクルを１０毎にすれば、近接ポリゴンペアの更新が早いので、正しい近接ポリゴンペアＰ３１,Ｐ３２間の最近点距離結果ｄ３を計算して表示する。しかし、BubbleCollision法の実行回数が多くなり干渉チェックに時間がかかる。このように、試行錯誤的な更新サイクルの調整は難しい。

【００９９】そこで、Bubble Collision法を行う毎にその計算時間を測定し、その計算時間から次の更新サイクルを自動的に決定する方法を提案する。Bubble Collision法の計算時間の変動に対応して自動的にサイクルを決定することで、試行錯誤的でなく、計算時間の短縮と修正タイミングのバランスをうまく取れる見込みがある。このフローを図６２に示すが、図５８のフローに斜線部のステップ５６０１～５６０３を付加したものである。計算時間測定は、まずBubble Collision法を行う前に計算機システムコールにて計算機内部クロックで時間測定してｔ１を求める（ステップ５６０１）。Bubble Collision法を行った後に同様にしてｔ２を求め、Bubble Collision法の計算時間を次式

$t_{bc} = t2 - t1$

により計算する（ステップ５６０２）。

【０１００】そして、ポリゴンペアの更新サイクルを例えば次式

更新サイクル＝（Bubble Collision法の計算時間 t_{bc} × α(C)

により求める（ステップ５６０３）。但し、αは調整必要である。従って、Bubble Collision法の干渉チェックに１００ｍｓかかり、連続型Bubble Collision法に１ｍｓかかり、α＝１．５とすると更新サイクルは１５０で平均計算時間は１．６６ｍｓ(=(100+149･1)/150)程度になる。図６３(a)では、１回目（ａ）より２回目（ｂ）のBubble Collision法の計算時間が長くなっているので、更新サイクルが長くなる。３回目（ｃ）のBubble Collision法の計算時間は短いので、更新サイクルは短くなる。平均計算時間は点線のように多少変動するが、Bubble Collision法のみの時に比べて小さくなる。このように、計算時間 t_{bc} が長くなるほど更新サイクルを長くすることにより、平均計算時間を短くでき、しかも、クイックレスポンスが可能になる。ただし、間違った結果を出力する可能性があるが、近接ポリゴンペはBubble collision法により更新され、正しい結果を出力できるようになる。

【０１０１】以上では、t_{sc}が大きいほど更新サイクルを長くした場合であるが、厳密に正しい出力結果が必要の場合には、t_{sc}が大きいほど更新サイクルを短くする。図６３(b)では、１回目ａより２回目ｂのBubble Collision法の計算時間が長くなっているので、更新サイクルは短くなる。３回目ｃのBubble Collision法の計算時間は短いので、更新サイクルは長くなる。更新サイクルにより平均計算時間は多少変動するが、Bubble Collision法のみの時に比べて遥かに小さくできる。又、t_{sc}が大きいとポリゴンペアの更新サイクルが短くなって平均計算時間が長くなるが、厳密に正しい近接ポリゴン間の干渉チェックを行うことができ、正しい最近点、最近点間距離を出力できる。以上のように、Bubble Collision法を行う毎にその計算時間を測定し、計算時間の変動に対応して自動的に更新サイクルを決定することで、試行錯誤的でなく、計算時間の短縮と修正タイミングのバランスをうまく取れる。

【０１０２】（Ｇ）連続型Bubble Collision法の第１変形例

連続型Bubble Collision法は移動距離／姿勢変化が小さいと仮定するので、大移動が発生すると、間違ったポリゴンペア間の距離を示すのが目立つ（図６１(b)）。移動距離／姿勢変化があるしきい値より大きい場合には、近接ポリゴンペアの更新サイクル（図６３中のｂ、ｃ）に違しなくてもBubble Collision法による干渉チェックを行い、近接ポリゴンペアを更新する。これにより、正しい干渉チェック結果を得ることができる。例えば、

$|\Delta x| > |\Delta x_{limit}|$

ならば、Bubble Collision法による干渉チェックを行い、近接ポリゴンペアを更新する。図６４はかかる処理のフローであり、斜線部のステップ５６０４を図６２のフローに付加したものである。上式中、Δxは１サイクルの間の物体移動量であり、Δx_{limit}はしきい値である。

【０１０３】しきい値Δx_{limit}は、図６５(a)～(c)に示すように、移動物体Ｂの最大辺の長さをL_{max}とすれば、L_{max}/βにより、あるいは、移動物体の最上位包絡球の半径をR_0とすれば、R_0/βにより、あるいは、移動物体のLeaf球の最大半径をR_{Lmax}とすれば、$R_{Lmax}\cdot\beta$により決定する（βは２～４程度で調整必要）。又、しきい値Δx_{limit}は、干渉チェック対象中の最小物体の最大辺の長さをL_{max}とすればL_{max}/βにより、あるいは、干渉チェック対象中の最小物体の最上位包絡球の半径をR_0とすればR_0/βにより、あるいは、干渉チェック対象中の最小物体のLeaf球の最大半径をR_{Lmax}とすれば$R_{Lmax}\cdot\beta$により決定することもできる。

【０１０４】（Ｈ）連続型Bubble Collision法の第２変形例

連続型Bubble Collision法は移動距離／姿勢変化が小さいと仮定するので、大移動が発生すると、間違ったポリゴンペア間の距離を示すのが目立つ（図６１(b)）。移動速度があるしきい値より大きい場合には、近接ポリゴンペアの更新サイクル（図６３中のｂ、ｃ）に違しなくてもBubble Collision法を行い、近接ポリゴンペアを更新する。これにより、正しい干渉チェック結果を得ることができる。例えば、

$|v| > |v_{limit}|$

ならば、Bubble Collision法を行い、近接ポリゴンペを更新する。図６６はかかる処理のフローであり、斜線部のステップ５６０５を図６２のフローに付加したものである。上式中、ｖは物体移動速度、v_{limit}はしきい値である。

【０１０５】しきい値v_{limit}は、移動物体の最大辺の長さL_{max}とすれば、$L_{max}/\beta\cdot dt$により、移動物体の最上位包絡球の半径をR_0とすれば、$R_0/\beta\cdot dt$により、あるいは、移動物体のLeaf球の最大半径を$R_{L\,max}$とすれば、$R_{L\,max}\cdot\beta/dt$により決定する（但し、ｄｔは積分きざみ、βは２、３、４程度で調整必要）。又、しきい値v_{limit}は、干渉チェック対象中の最小物体の最大辺の長さをL_{max}とすれば$L_{max}/\beta\cdot dt$により、あるいは、干渉チェック対象中の最小物体の最上位包絡球の半径をR_0とすれば、$R_0/\beta\cdot dt$により、あるいは、干渉チェック対象中の最小物体のLeaf球の最大半径をR_{Lmax}とすれば$R_{Lmax}\cdot\beta/dt$により決定することもできる。

【０１０６】（Ｉ）連続型Bubble Collision法の第３変形例

連続型Bubble Collision法は移動距離／姿勢変化が小さいと仮定するので、大移動が発生すると、間違ったポリゴンペア間の距離を示すのが目立つ（図６１(b)）。最近点距離比があるしきい値より大きい場合には、近接ポリゴンペアの更新サイクル（図６３中のｂ、ｃ）に違しなくてもBubble Collision法を行い、近接ポリゴンペアを更新する。これにより、正しい干渉チェック結果を得ることができる。例えば、（最近点距離比）＞γ（但し、γは３～５程度）ならば、Bubble Collision法を行い、近接ポリゴンペアを更新する。最近点距離が前回の結果の５倍になるということは、大移動が発生したと考えられるからである。

【０１０７】ある近接ポリゴンペアについて、

（最近点距離比）＝｜（今回の最近点距離）／（前回の最近点距離）｜

とする。例えば図６７では近接ポリゴンペアＰ１１，Ｐ１２については｜ｄ１′／ｄ１｜、近接ポリゴンペアＰ２１，Ｐ２２では、｜ｄ２′／ｄ２｜である。γ＝３とすると｜ｄ１′／ｄ１｜＞３、｜ｄ２′／ｄ２｜＞３になるので、BubbleCollision法を行い、近接ポリゴンペアを更新して干渉チェックを行い、正しい最近点距離ｄ３を出力する。

【０１０８】図６８はかかる処理のフローであり、斜線部のステップ５６０６　～５６０７を図６２のフローに付加したものである。ステップ５６０６では、修正フラグBCmodeの判定を行う。BCmodeが１ならばBubble Collision法を行い、近接ポリゴンを更新して干渉チェックを行う。BCmodeが０ならば、近接ポリゴンペアの干渉チェックを行う。ステップ５６０７では、まず、最近点距離ｄｉ′の保存と、その時の近接ポリゴンペアの保存を行う。次に最近点距離比計算で｜ｄｉ′／ｄｉ｜を計算する。（ｄｉは一つ手前のサイクルの状態での最近点距離である）。最近点距離比判定で｜ｄｉ′／ｄｉ｜＞γ（但し、γは３～５程度）ならば、修正フラグBCmode＝１とする。そうでなければ、修正フラグBCmode＝０とする。修正フラグBCmode＝１ならば、次のサイクルでBubble Collision法を行い、近接ポリゴンを更新する。尚、第１～第３変形例は、図６２に示す連続型Bubble collision法の変形例として説明したが、図５８の連続型Bubble collision法を同様に変形することができる。

【０１０９】（Ｊ）物体静止の干渉チェック効率化

(a) 凸包外における物体静止の干渉チェック効率化

従来は、高速化を図ることでリアルタイムな干渉チェックを可能にすることに重点を置いている。そして、リアルタイム性が保てるということでステップ毎に毎回、干渉チェックを行っている。しかし、干渉チェックを一度行った後に、チェック対象物体がすべて静止していれば、前回の結果と同じであるので、その結果を更新する必要はない。そこで、速度指令、位置変位を監視し、速度指令あるいは位置変位が零ならば、前サイクルの干渉チェック結果をそのまま使って計算を省略する。このようにすれば、干渉チェック対象物体グループが二つ以上の場合に効果が大きい。なぜならば、移動物体を含むグループのみの干渉チェック計算だけで良いからであり、干渉チェックの効率化が図られる。

【０１１０】一例として、図６９のような自動車工場内ロボットの干渉チェックを考えてみる。ロボット１とロボット２はプログラミングされた作業軌道で塗装作業等を行い、ロボット３はマウス等のＭＭＩによってインタラクティブに指令値が与えられ、タイヤのネジ締め作業等を行っているとする。この場合、干渉チェック対象物体のグループとして３つ存在する。第１グループはロボット１と車、第２グループはロボット２と車、第３グループはロボット３とロボット２と車である。各グループの干渉チェック計算時間が１０ｍｓずつかかると仮定すると次のような効率化が可能である。

【０１１１】1) ロボット１が静止の場合

第１グループの干渉チェックデータが維持され、総計算時間は２０ｍｓとなる

2) ロボット２が静止の場合

第２グループの干渉チェックデータが維持され、総計算時間は２０ｍｓとなる。　3) ロボット１、ロボット２が静止の場合

ロボット１，ロボット２の干渉チェックデータが維持され、総計算時間は１０ｍｓとなる。この考えは車が他の装置、製品であっても、ロボットが多数の場合にも拡張でき、干渉チェックの効率化を図れる。

【０１１２】（b） 凸包内における物体静止時の干渉チェック効率化

以上は凸包外における物体静止時の場合であるが、それを物体がかなり接近して、凸包内にある場合にも適用できる。ただし、若干の相違がある。凸包内では連続型Bubble Collision法による干渉チェックを行っているが、干渉チェックを一度行った後に、チェック対象物体がすべて静止していれば、前回の結果を更新する必要はない。そこで、速度指令、位置変位を監視し、これらが零ならば、前サイクルの干渉チェック結果をそのまま使って干渉チェック処理を省略する。但し、静止後の近接ポリゴンペアの更新サイクルで一度だけBubble Collision法を行って正しい結果を求める必要がある。このようにすれば、干渉チェック対象物体グループが二つ以上の場合に、移動物体を含むグループのみの干渉チェック計算だけで良いので効果が大きい。

【０１１３】例えば、図７０に示す凹型物体Ｃと立方体Ｄの簡易なモデルについて説明する。移動中ｔ１では（ａ）の状態にあり、正しく近接ポリゴンペアＰ１１，Ｐ１２間の干渉チェックが行われる。しかし、その後の移動量が大きく時刻ｔ２で静止すると（ｂ）の状態になり、この状態で間違ったポリゴンペアＰ１１，Ｐ１２間で干渉チェックを行って誤った結果を示す。そこで、速度指令、位置変位が零において、単に前サイクルでの干渉チェック結果をそのまま使って計算を省くという方法では、再移動後の時刻ｔ５までは誤ったポリゴンペアＰ１１，Ｐ１２が保存し続けられる。これを避けるためには、時刻ｔ２で静止した後、時刻ｔ３の近接ポリゴンペアの更新サイクルで一度だけBubble Collision法を行って、正しい近接ポリゴンペアＰ２１，Ｐ２２を求め、これらの間の干渉チェックを行い、その結果を保持し、時刻ｔ４まではその干渉チェック結果をそのまま使って干渉チェック処理を省略する。これを図６９のような自動車工場内ロボットの干渉チェックに拡張すると、物体がかなり接近して、凸包内にある場合の干渉チェックの効率化に結び付く。

【０１１４】（Ｋ）産業上の応用分野

本発明は、以下のような分野における応用が可能である。

(a) 機構設計用ＣＡＤシステム

機構設計では、製品を組み立てた時に不用意な干渉が度々発生する。このようなことを事前に防ぐためには、機構設計用ＣＡＤシステム上にて部品間の接合関係、マージンを入念にチェックしておくことが重要になる。本発明のリアルタイム干渉チェック方法を機構設計用ＣＡＤ

システムに組み込むことにより、実際の製品に近い形での事前チェックが可能となる。

【０１１５】(b) 移動ロボットの経路計画

マニピュレータや自走車などの移動ロボットでは、他の物体と衝突しないよう経路計画を事前に行った後に駆動させる場合が多い。例えば、マニピュレータを使った部品組立て工程では、機構設計用ＣＡＤシステム上に部品モデル、マニピュレータモデルを持ち、マニピュレータを使って各部品を組み立てていく工程をプランニングすることがある。このような場合に部品間の干渉、部品とマニピュレータ間との干渉をチェックしながら経路計画することが必要となる。また、掃除ロボット、警護ロボット、運搬ロボットなどの自走ロボットでは、地点間の移動において壁や柱または机等の障害物との衝突を回避した経路計画が必要である。これら経路計画において本発明の干渉チェック方法を適用して、干渉の事前チェックすることができる。

【０１１６】(c) マルチメディアにおけるアニメーション作成

マルチメディアでは、コンピュータグラフィックス、実画像、音声などをミックスさせた様々な応用が期待されており、又、コマーシャルや映画の分野ではコンピュータグラフィックスを使ったリアリティの高いアニメーション作成が望まれている。従来のコンピュータグラフィックスの世界では、アニメーションはもっぱらオフライン的に作成されており、干渉問題も総当たり法を使った方法が取られていた。これからのアニメーション作成は、高速なグラフィックコンピュータを使ったリアルタイム／インタラクティブ作成が主流になり、その場合にはＣＧモデル間の干渉を如何に高速に解くかが鍵となる。例えば、アンドロイドモデル（人間の形をしたＣＧモデル）を歩かせる場合には、アンドロイドモデルと地上との接触問題を解かなければならず、また、自動車の衝突をシミュレーションする場合には、まさしく自動車間の干渉問題を解く必要がある。本発明の高速の干渉チェック方法は、リアルタイム／インタラクティブアニメーション作成において必須技術となるものである。

【０１１７】(d) ゲームソフト

従来のゲームソフトはもっぱら２次元的なものが多かったが、今後は、グラフィックコンピュータを使い、３次元的に表示するものが主流になると予想される。ゲームソフトでは、例えば、シューティングゲームにおけるミサイルと戦闘機の衝突、レーシングゲームにおける車同志の衝突など物体間の干渉問題が多くの場面に絡んでおり、本発明の干渉チェック方法は、これらに対して非常に強力な手段を提供するものである。以上、本発明を実施例により説明したが、本発明は請求の範囲に記載した本発明の主旨に従い種々の変形が可能であり、本発明はこれらを排除するものではない。

【０１１８】

【発明の効果】本発明によれば、高速に非凸多面体間の干渉チェックを行うことができる。本発明によれば、干渉チェックの前処理である凸包の生成を効率良く、高速に行って干渉チェックを行うことができる。

【図面の簡単な説明】

【図１】本発明の原理説明図である。

【図２】非凸多面体の説明図である。

【図３】凸包の説明図である。

【図４】干渉チェックのシステム構成図である。

【図５】干渉チェック前処理の概略フローである。

【図６】多面体を表現するためのデータ構造説明図である。

【図７】多面体を表現するためのＣ言語によるデータ構造である。

【図８】凸包構成アルゴリズムである。

【図９】Ｃ言語による２次元凸包構成基本関数である。

【図１０】一次元凸包構成説明図である。

【図１１】アッパーブリッジ、ロワーブリッジの構成法説明図である。

【図１２】Ｃ言語による２次元凸包マージ関数の構成である。

【図１３】凸包構成の説明図である。

【図１４】凸包構成の説明図である。

【図１５】凸包構成の説明図である。

【図１６】凸包構成の説明図である。

【図１７】Ｃ言語による３次元凸包マージ関数の構成である。

【図１８】３次元凸包構成の説明図である。

【図１９】ラッピングプロセス説明図である。

【図２０】三角形ポリゴンのパッチングプロセスの説明図である。

【図２１】Ｃ言語による三角形パッチング関数である。

【図２２】バックアップ凸包構成アルゴリズムにおけるラッピング及びパッチングプロセス説明図である。

【図２３】近接点線形リスト説明図である。

【図２４】凸要素毎のポリゴンデータの説明図である。

【図２５】近接点線形リスト構成アルゴリズムの説明図である。

【図２６】近接点線形リストのダンプファイルフォーマットである。

【図２７】リーフ球の説明図である。

【図２８】三角形ポリゴン説明図である。

【図２９】辺上のリーフ球による被覆説明図である。

【図３０】三角形ポリゴン内部の被覆説明図である。

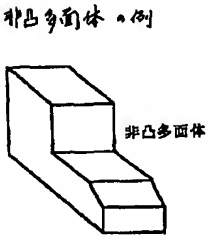
【図３１】Ｃ言語による２分木ツリー用のデータ構造である。

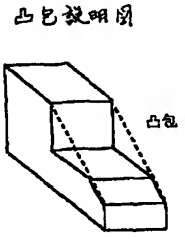
【図３２】２分木ツリー構成のフローである。

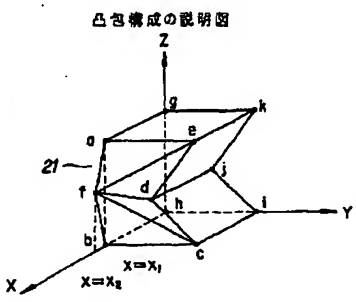
【図３３】２分木ツリーの例である。

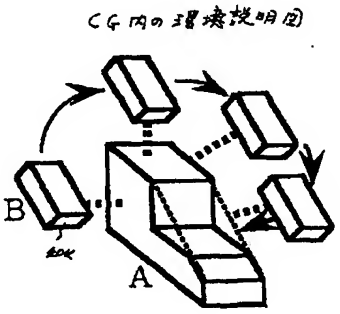
【図３４】Ｃ言語によるバブルコライジョンチェック基本関数（その１）である。

【図３５】Ｃ言語によるバブルコライジョンチェック基本関数（その２）である。
【図３６】球間距離説明図である。
【図３７】Ｃ言語による干渉チェックデータ構造（その１）である。
【図３８】Ｃ言語による干渉チェックデータ構造（その２）である。
【図３９】Ｃ言語による干渉チェックデータ構造（その３）である。
【図４０】Gilbert法の基本的な説明図である。
【図４１】連続型Gilbert法の説明図である。
【図４２】融合アルゴリズムのフローである。
【図４３】(2+1)次元干渉チェック説明図である。
【図４４】メタツリー説明図である。
【図４５】メタツリー構成を包含した連続型干渉チェックのフローである。
【図４６】メタツリー構成を包含した連続型干渉チェックの別のフローである。
【図４７】干渉チェックの全体フローである。
【図４８】ＣＧ内の環境説明図である。
【図４９】マウスを用いた三次元入力方式のシステム構成図である。
【図５０】三次元座標入力部の説明図である。
【図５１】リーフ球の作成説明図である。
【図５２】三角形ポリゴン上のリーフ球説明図である。
【図５３】階層包絡球の２分木ツリー構造説明図である。
【図５４】階層包絡球の２分木ツリー構造の三次元イメージ説明図である。
【図５５】階層包絡球の構成例（直方体の場合）である。
【図５６】階層包絡球の構成例（非凸多面体の場合）である。
＊【図５７】連続型バブルコライジョン法説明図である。
【図５８】連続型バブルコライジョン法のフロー図である。
【図５９】近接球ペアの探索／近接ポリゴンペアの探索説明図である。
【図６０】連続型バブルコライジョン法の計算時間説明図である。
【図６１】近接ポリゴンペア更新説明図である。
【図６２】連続型バブルコライジョン法の更新サイクル自動決定法のフローである。
【図６３】連続型バブルコライジョン法の計算時間説明図である。
【図６４】連続型バブルコライジョン法の修正方法のフローである。
【図６５】物体移動量のしきい値の決定法説明図である。
【図６６】連続型バブルコライジョン法の修正方法のフローである。
【図６７】最近点距離比による修正説明図である。
【図６８】連続型バブルコライジョン法の修正方法のフローである。
【図６９】自車工場内ロボット干渉チェック説明図である。
【図７０】凸包内における物体静止時の干渉チェックの効率化の説明図である。
【符号の説明】
１・・凸包構成部
２・・近接点線形リスト作成部
３・・階層包絡球２分木ツリー生成部
４・・記憶部
５・・凸包間干渉チェック部
６・・バブルコライジョン法実行部
７・・ポリゴンペア間干渉チェック部

【図２】


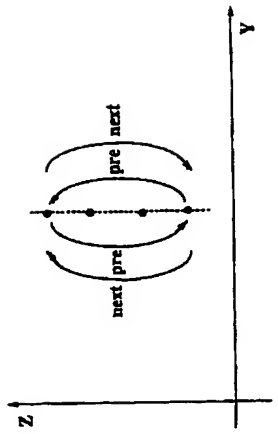
【図３】


【図１３】


【図４８】


【図１０】

１次元凸包構成



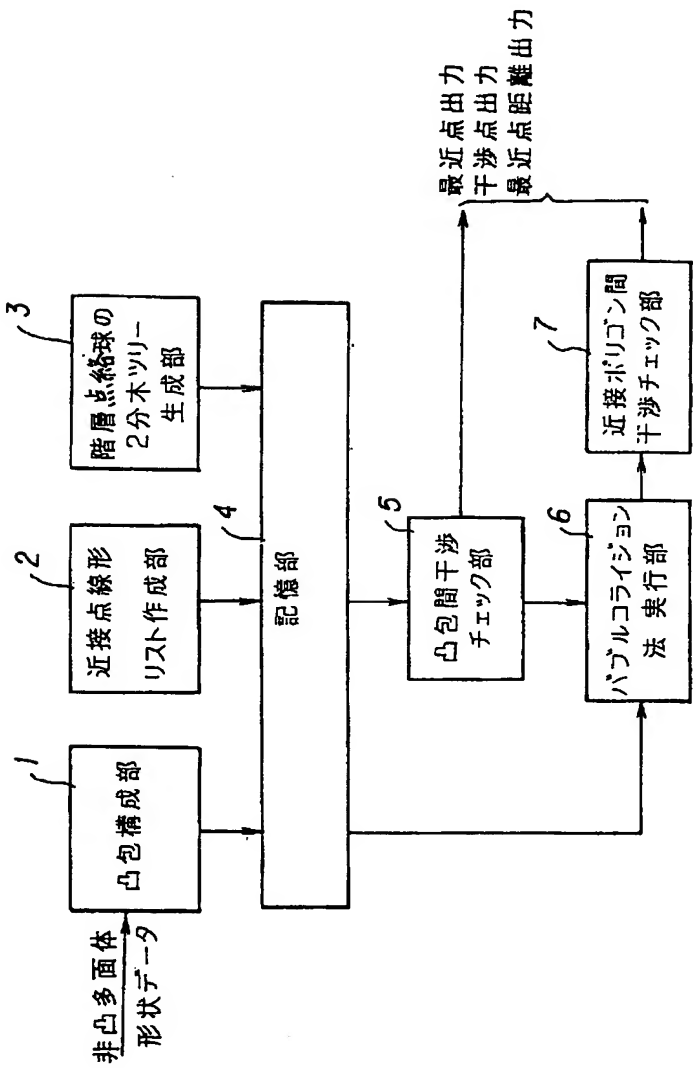
【図３１】

Sphere Tree 用データ構造

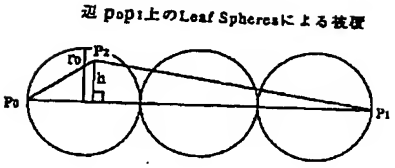
```
class Leaf{
  Vector3 center;
  Scalar r;
  Leaf *next;
  Plane *plane;
};

class Leaves{
  Leaf *leaves;
  Vector3 max;
  Vector3 min;
  Vector3 avcenter;
  int leafcnt;
};

class Sphere{
  Vector3 center;
  Scalar r;
  Sphere *left;
  Sphere *right;
  Leaves *leaves;
};
```

【図１】

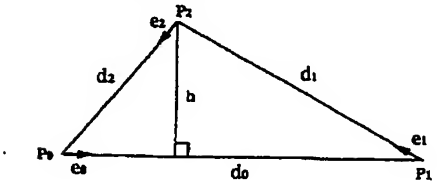
本発明の原理説明図



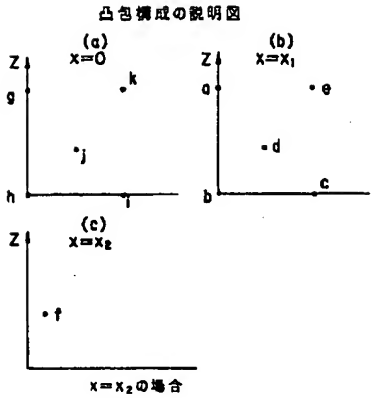
【図２９】



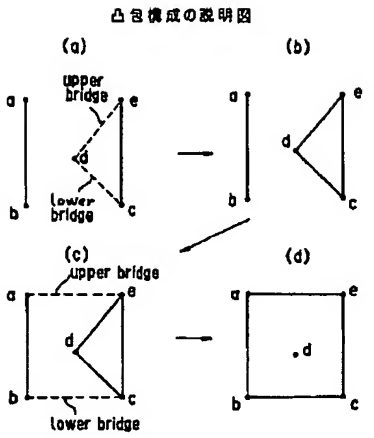
【図２８】



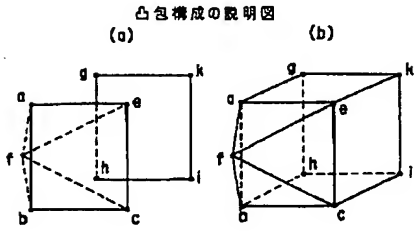
【図１４】



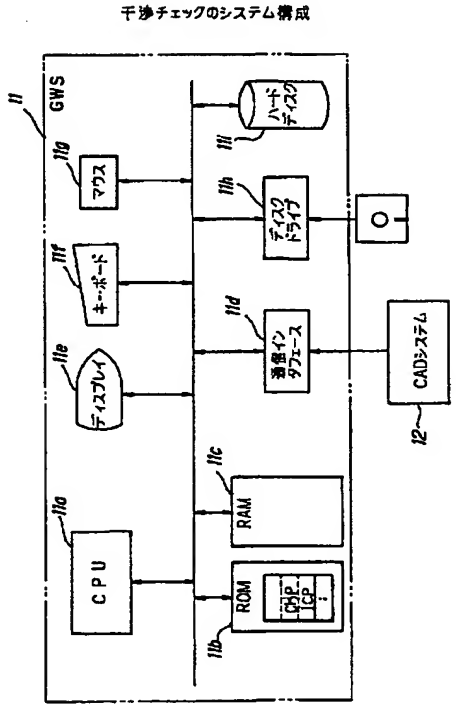
【図１５】



【図１６】



【図４】

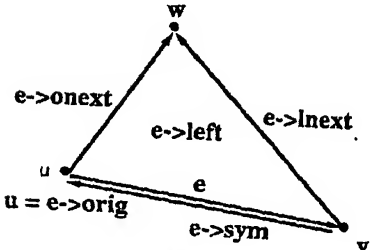


【図6】

多面体を表現するためのデータ構造説明図

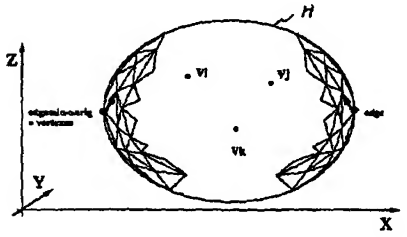
(a)

三角形パッチ



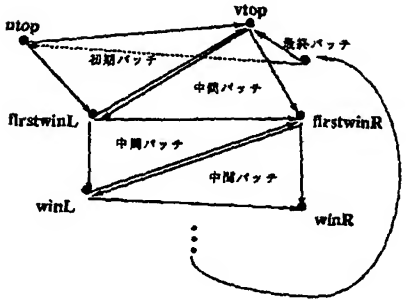
(b)

凸包 H



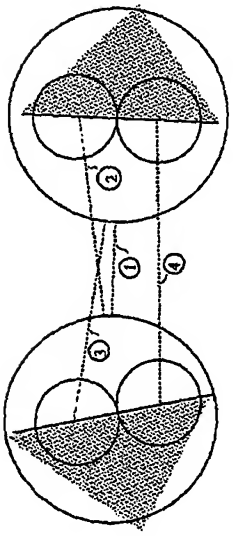
【図20】

三角形ポリゴンのPatching process



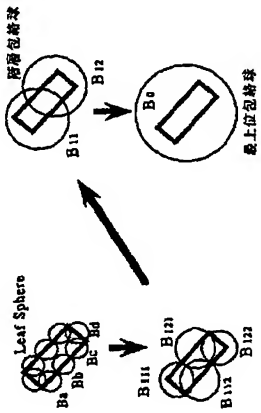
【図36】

Sphere間の距離



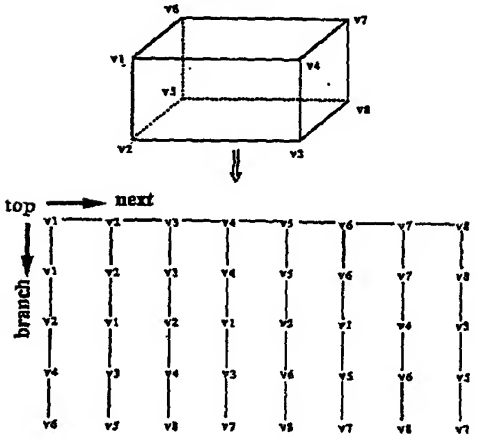
【図55】

階層包絡球の構成例（直方体Bの場合）



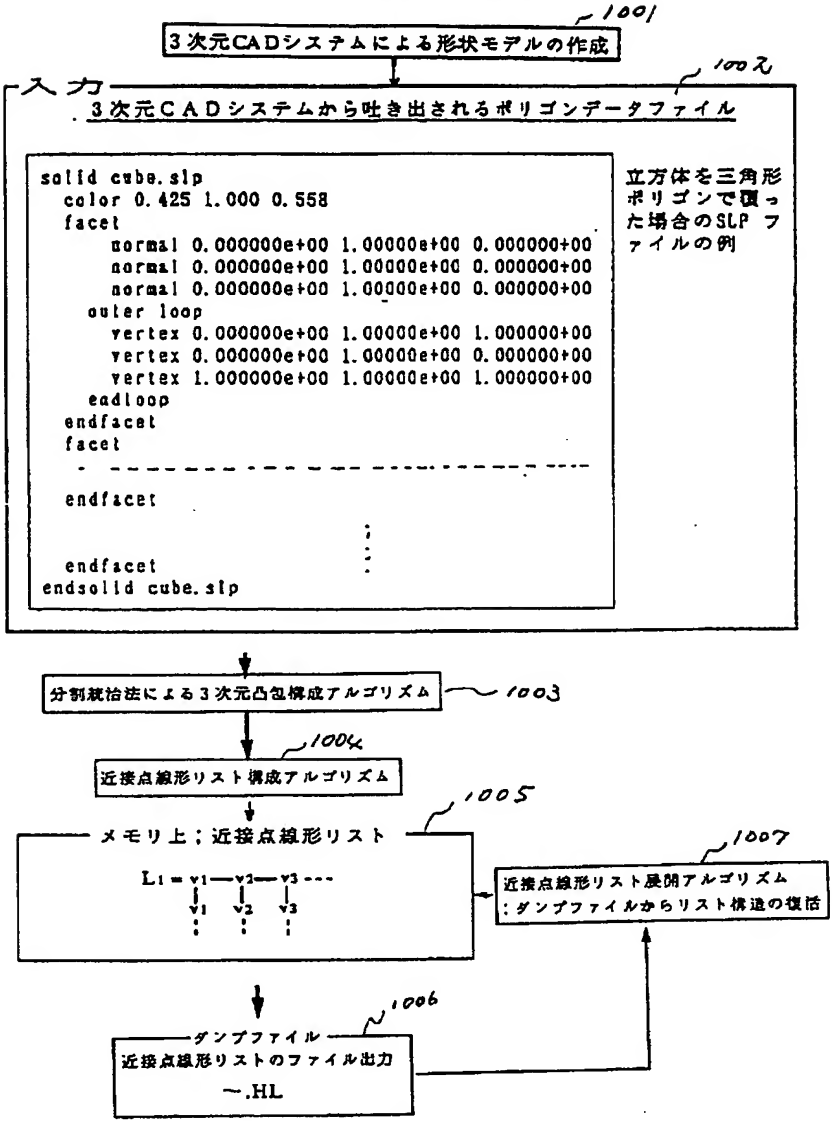
【図23】

近接点線形リストの例



【図５】

干渉チェック前処理の概略フロー



【図7】

多面体を表現するためのデータ構造

```
Class Facet{
      int color;
      int searchflg;
};

Class Vertex{
      int color;
      Vertex *next;
      Vertex *pre;
        // 頂点座標
      Vector3 v;
        // perturbation of v
      Vector3 dv;
};

Class Edge{
      int color;
      Apex *orig;
      Facet *left;
      Edge *sym;
      Edge *onext;
      Edge *lnext;
};

Class Convex{
      int side;
      int vcnt;
      Vertex *vertexes;
      Eedge *edge;
      Eedge *edgemin;
      Vertex *maxX;
      Vertex *minX;
      Vertex *maxY;
      Vertex *minY;
};

Class List{
      Edge *u;
      Edge *v;
      Edge *win;
      Vertex *eu;
      Vertex *ev;
      int side;
      List *next;
};
```

【図9】

2次元凸包構成基本関数

```
MakeConvex2Dim(con){
    if(maxY!= null) {maxY = con->CalcMaxY();};
    if(minY != null) {minY = con->CalcMinY();};
    if( | maxY->v(1) - minY->v(1) | < ε ){
          MakeConvex1Dim(con);
          return;
    }
    Convex *conLeft = new Convex;
    Convex *conRight = new Convex;

    con->DivideY(conLeft, conRight);
    MakeConvex2Dim(conLeft);
    MakeConvex2Dim(conRight);
    Merge2Dim(conLeft, conRight);
    return;
}
```

//モデルデータconに対し、Y軸方向のmin,maxを計算

//min,maxの差が微小量εより小さければ1次元の凸包構成を行う

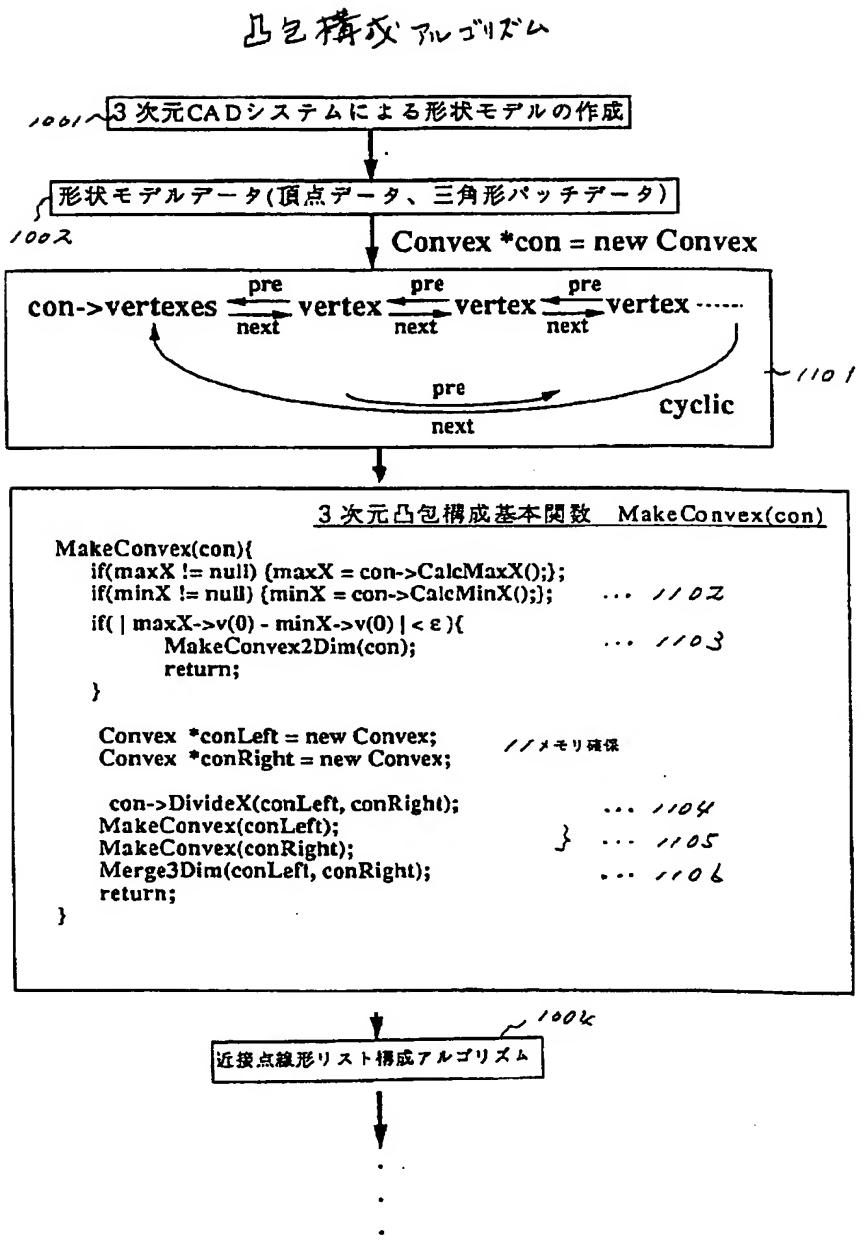
//メモリ確保

//conをY座標値の中心値を境にconLeft, conRightに分割

//conLeft, conRight各々に対して再帰的に凸包構成

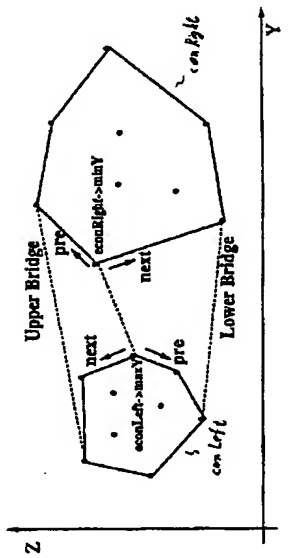
//凸包conLeft, conRightをマージ

【図８】


凸包構成アルゴリズム
1001
3次元CADシステムによる形状モデルの作成
形状モデルデータ(頂点データ、三角形パッチデータ)
1002
Convex *con = new Convex
con->vertexes
pre
next
vertex
vertex
vertex
cyclic
1101
3次元凸包構成基本関数 MakeConvex(con)
MakeConvex(con){
if(maxX != null) {maxX = con->CalcMaxX();};
if(minX != null) {minX = con->CalcMinX();};
1102
if(| maxX->v(0) - minX->v(0) | < ε){
MakeConvex2Dim(con);
1103
return;
}
Convex *conLeft = new Convex;
Convex *conRight = new Convex;
//メモリ確保
con->DivideX(conLeft, conRight);
1104
MakeConvex(conLeft);
MakeConvex(conRight);
1105
Merge3Dim(conLeft, conRight);
1106
return;
}
1004
近接点線形リスト構成アルゴリズム

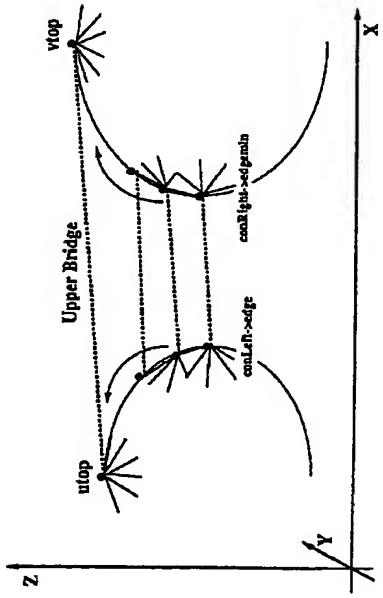
【図１１】

Merge2Dim(,)におけるUpper Bridge, Lower Bridgeの構成法



【図１８】

MakeBridge3Dim(conLeft, conRight)



【図１２】

Merge2Dim(conLeft, conRight)の構成

```
Merge2Dim(conLeft, conRight){
    vlup = conLeft->maxY;          // Upper bridge 用初期値設定
    vrup = conRight->minY;
    vldown = conLeft->maxY;        // Lower bridge 用初期値設定   ・・・(30)
    vrdown = conRight->minY;

    do{                                                  // Upper bridge を見つける  ・・・(31)
        RC0 = vlup->next->ConvexJudgeUp2Dim(vlup,vrup); // 頂点vlup->nextとLine(vlup,vrup)の上下関係を判定
        RC1 = vrup->pre->ConvexJudgeUp2Dim(vlup,vrup);  // 頂点vrup->preとLine(vlup,vrup)の上下関係を判定
        if(RC0 > 0.0) vlup = vlup->next;    // もしvlup->nextがLine(vlup,vrup)の上にあるならばvlupをnext方向に更新
        if(RC1 > 0.0) vrup = vrup->pre;     // もしvrup->preがLine(vlup,vrup)の上にあるならばvrupをpre方向に更新
    }while(RC0 > 0.0 || RC1 > 0.0);         // vlup->next, vrup->preがLine(vlup,vrup)の下にくるまで繰り返す

    do{                                                  // 例題にしてT1,lower bridge を見つける・・・(32)
        RC0 = vldown->pre->ConvexJudgeUp2Dim(vldown,vrdown);
        RC1 = vrdown->next->ConvexJudgeUp2Dim(vldown,vrdown);
        if(RC0 < 0.0) vldown = vldown->pre;
        if(RC1 < 0.0) vrdown = vrdown->down;
    }while(RC0 < 0.0 || RC1 < 0.0);
}
```

【図１７】

Merge3Dim(conLeft, conRight)の構成

```
Merge3Dim(conLeft, conRight){
 // Phase 0
  if(conLeft->vcnt <= 3 && conRight->vcnt <= 3){
      MakeConvexLowNumber(conLeft, conRight);
      return;
  }

  if( 1 <= conLeft->vcnt <= 3 ){
      MakeConvexBackL(conLeft, conRight);
      return;
  }

  if( 1 <= conRight->vcnt <= 3 ){
      MakeConvexBackR(conLeft, conRight);
      return;
  }

 // Phase 1
    MakeBridge3Dim(conLeft, conRight);
 // Phase 2
    Wrap(utop, vtop);
 // Phase 3
    PatchTriangles(L);
 // Phase 4
    L->ResetColor();
    edge = MaxEdge(conRight->edge);
    edgemin = MinEdge(conLeft->edgemin);
    maxX = conRight->maxX;
    minX = conLeft->minX;
    delete L;
    return;
  }
```

// 頂点数が６以下におけるダイレクトなマージング

// conLeftの頂点数が３以下におけるバックアップ
アルゴリズムによる凸包構成

// conRightの頂点数が３以下におけるバックアップ
アルゴリズムによる凸包構成

// Upper Bridge の構成

// Wrapping process

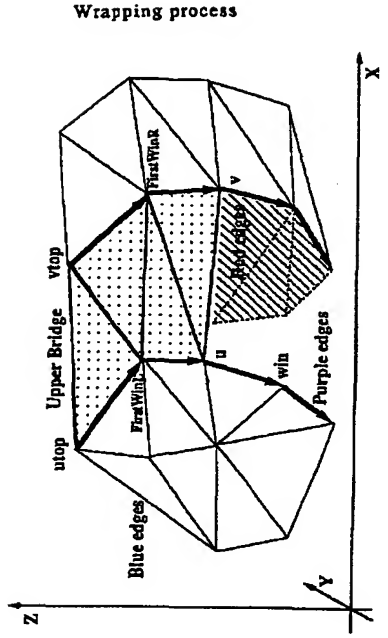
// 三角形ポリゴンのPatching process

// Purple edges のレセッティング

// 最大エッジ edge, 最小エッジ edgeminの設定

// maxX, minXの設定

【図１９】

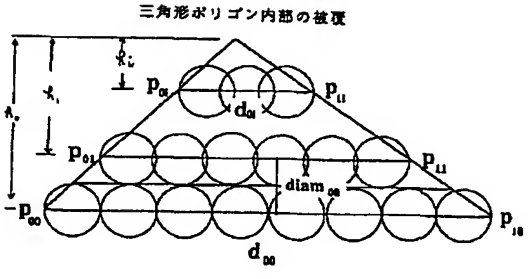


【図２１】

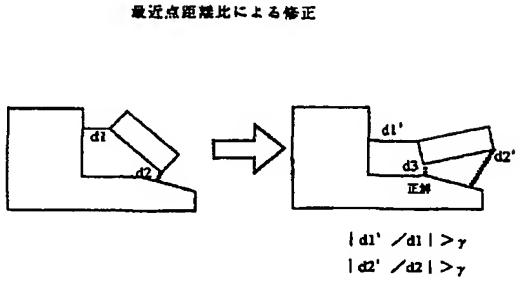
三角形パッチング関数

```
PatchTriangles(L){
  Edge *edumy;
  List *ldumy;
  if(L->side == LEFT){
      edumy = PatchTriangle(L->win->sym, L->v->orig);
      edtop = edumy->lnext->sym;
  }else if(L->side == RIGHT){
      edumy = PatchTriangle(L->win, L->u->orig);
      edumy = edumy->lnext;
  }                                    // 初期パッチの設定
  ldumy = L->next;

  while(ldumy->next){
    if(ldumy->side == LEFT){
        edumy = PatchTriangle(ldumy->win->sym, edumy);
    }else if(ldumy->side == RIGHT){
        edumy = PatchTriangle(edumy->sym, ldumy->win->sym);
    }
    ldumy = ldumy->next;               // 中間パッチの設定
  }

  if(ldumy->side == LEFT){
    PatchTriangleLast(ldumy->win->sym, edumy, edtop);
  }else if(L->side == RIGHT){
    PatchTriangleLast(edumy->sym, ldumy->win->sym, edtop);
  }                                    // 最終パッチの設定
}
```

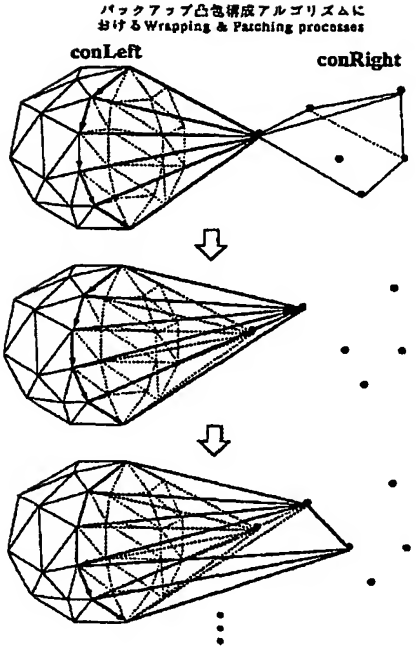
【図３０】



【図６７】

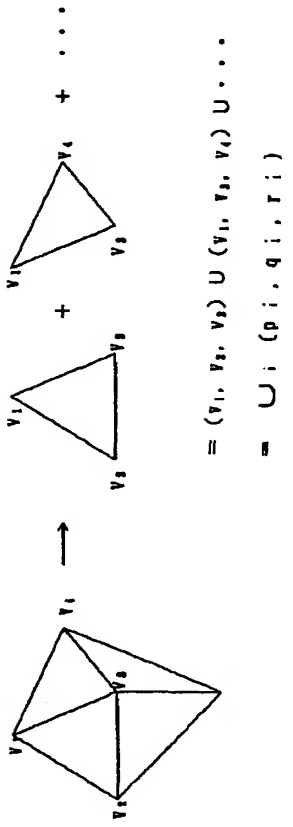


【図２２】



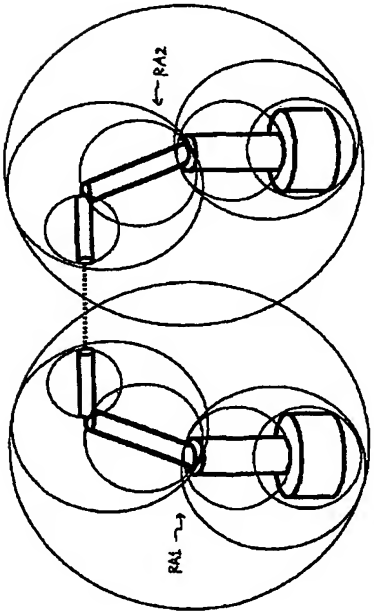
【図２４】

凸要素毎のポリゴンデータ説明図



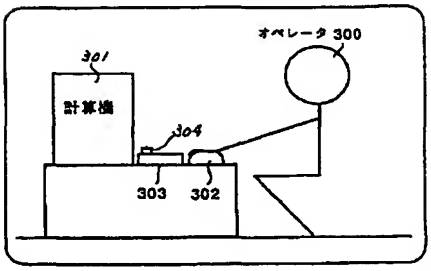
【図４４】

メタツリー



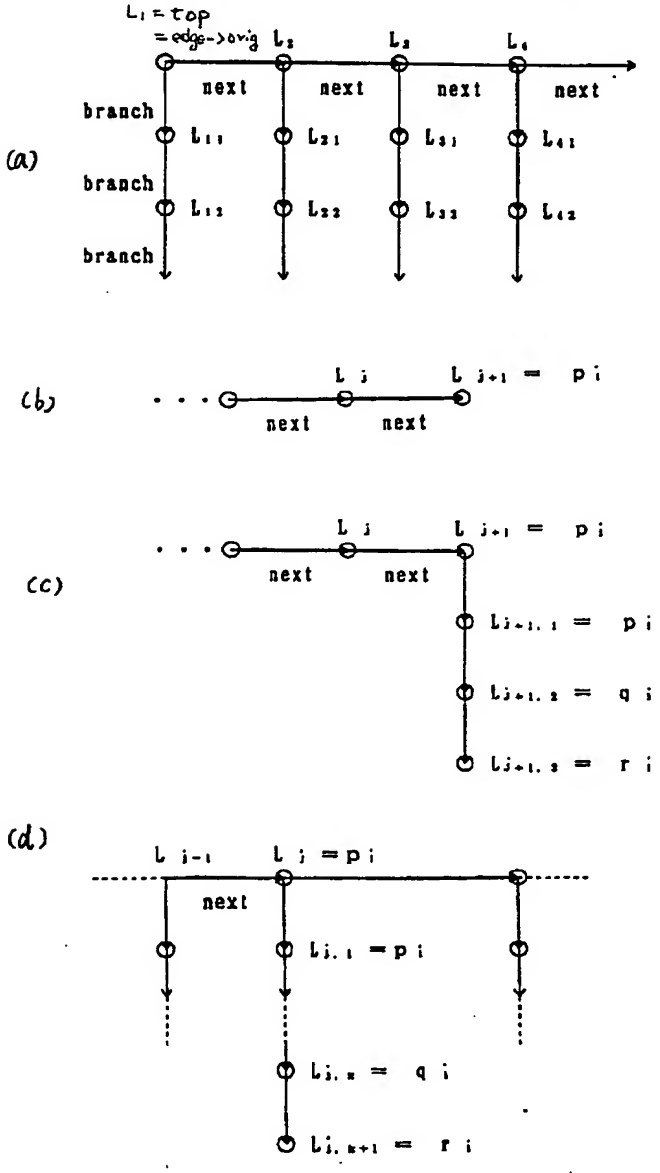
【図４９】

マウスを用いた三次元入力方式のシステム構成図



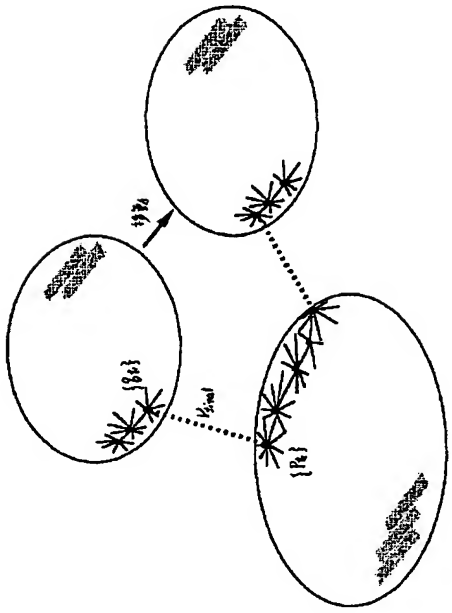
【図２５】

近接点線形リスト構成アルゴリズムの説明図



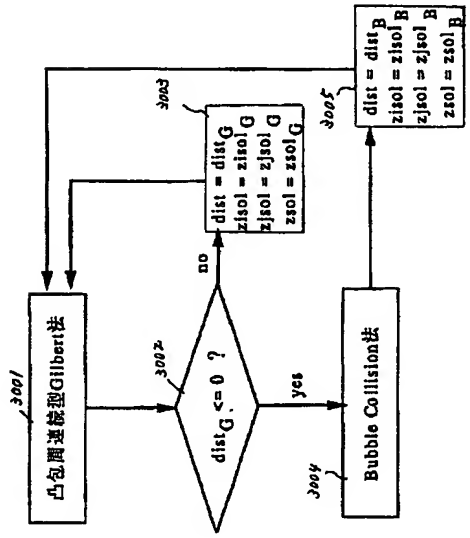
【図４１】

連続型Gilbert法



【図４２】

融合アルゴリズムのフロー

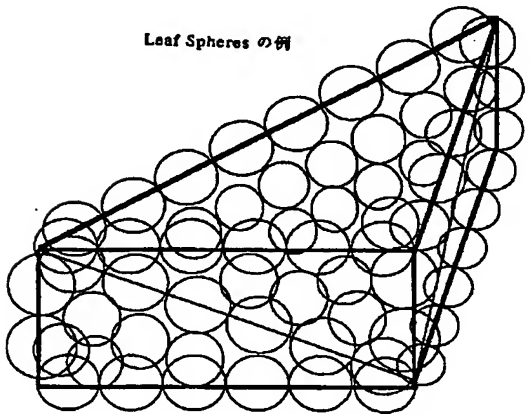


【図２６】

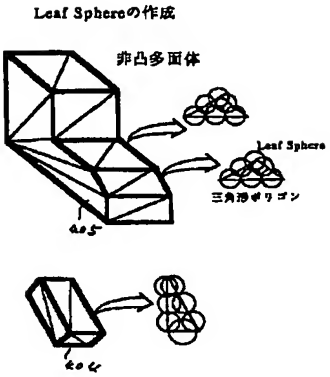
近接点線形リストのダンプファイルフォーマット

```
Convex      Dfund.L.0                                          //凸包名
    count 10                                                   //総頂点数
    R     1.224745e+00                                         //包絡球の半径
    Pc    1.00000e+00  5.000000e-01  5.000000e-01              //包絡球の中心
    com   1.00000e+00  5.000000e-01  6.000000e-01              //幾何中心
    max   2.00000e+00  1.000000e+00  1.000000e+00
    min   0.00000e+00  0.000000e+00  0.000000e+00
  Vertex                                                       //第０頂点
    pnum   0                                                   //頂点番号
    bcount  5                                                  //頂点の近接点個数
    vertex  0.00000e+00  1.000000e+00  1.000000e+00            //頂点座標
      Branch   0
      Branch   9                                               //近接点番号
      Branch   1
      Branch   7
      Branch   8
  EndVertex
  Vertex
  EndVertex                                                    //Vertexの繰返し
    ⋮
  Vertex
  EndVertex
EndConvex     Dfund.L.0                                        //ファイルエンド
```

【図２７】

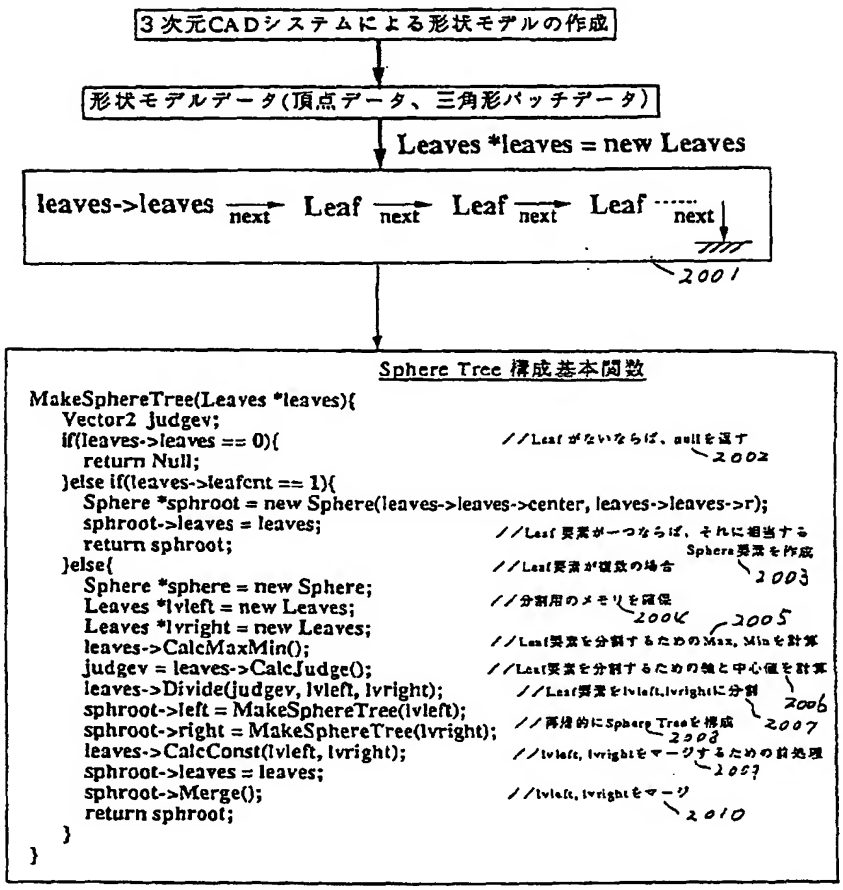


【図５１】



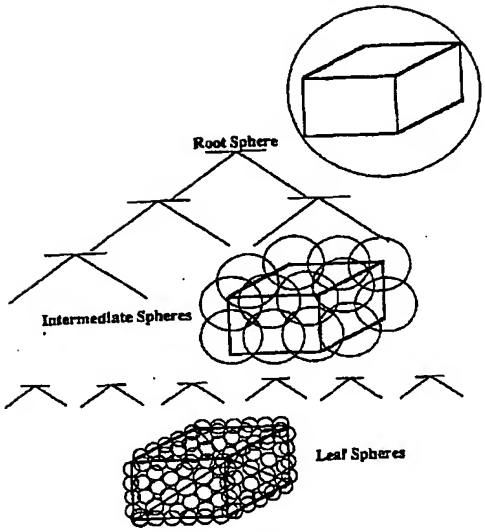
【図32】

Sphere Tree の構成フロー



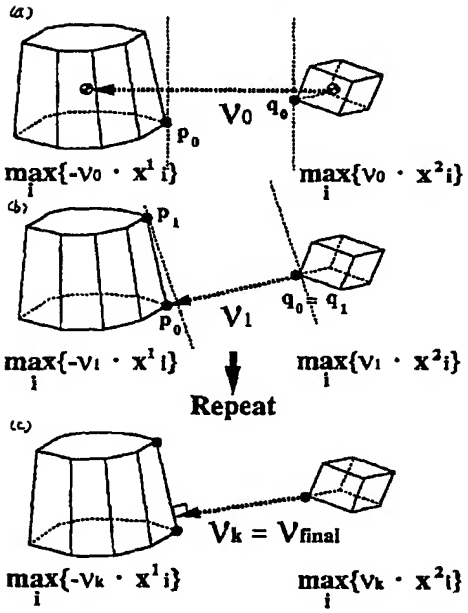
【図３３】

Sphere Tree の例



【図４０】

Gilbert法の基本的考え方



【図３７】

干渉チェックデータ構造（その１）

```
class ColQue{          // Bubble Collision用
  Sphere *spheres;       Sphere Tree
  int colflg;          // 衝突フラグ
  int contflg;         // 連続、非連続フラグ
  Vector3 zisol;       // 対象iの最近点
  Vector3 zjsol;       // 対象iの最近点
  Vector3 zsol;        // 最近ベクトル
  Scalar dist;         // 最近距離
  Matrix *R0;          // 座標変換マトリクス
  Vector3 *P0;         // 座標変換ベクトル
  ConvexHull *convex;  // 凸法
  Prims *vertex;       // 対象を構成する頂点集合
  Prims *v;            // vertexの座標変換
  ColQue *next;        // ColQueリスト用ポインタ
  ColQue *qcolq;       // 最近ColQueペア
};

class MetaLeaf{
  Vector3 center;
  Scalar r;
  MetaLeaf *next;
  int colqnum;
  ColQue *colq;
};

class MetaLeaves{
  MetaLeaf *metaleaves;
  Vector3 max;
  Vector3 min;
  Vector3 avcenter;
  int metaleafcnt;
};
```

続く

【図３４】

Bubble Collision Check基本関数 （その１）

```
BubbleCheck(Sphere *sp, Sphere *sptop, CollisionQue *cq){
  Scalar ldist, rdist, dist;
  Leaf *lf, *lftop;                                              ...2101
  if(cq->dist > SphereDist(sp, sptop)){
    if(sp->leaves->leafcnt > 1 && sptop->leaves->leafcnt > 1){
      if(sp->r > sptop->r){                                      ...2102
        if(sp->left != 0 && sp->right == 0){
          BubbleCheck(sp->left, sptop);
        }else if(sp->left == 0 && sp->right != 0){
          BubbleCheck(sp->right, sptop);
        }else{
          ldist = SphereDist(sp->left, sptop);
          rdist = SphereDist(sp->right, sptop);
          if(ldist < rdist){                                     ...2103
            BubbleCheck(sp->left, sptop);
            BubbleCheck(sp->right, sptop);
          }else{
            BubbleCheck(sp->right, sptop);                       ...2104
            BubbleCheck(sp->left, sptop);
          }
        }
      }else{                                                     ...2105
        if(sptop->left != 0 && sptop->right == 0){
          BubbleCheck(sp, sptop->left);
        }else if(sptop->left == 0 && sptop->right != 0){
          BubbleCheck(sp, sptop->right);
        }else{
          ldist = SphereDist(sp, sptop->left);
          rdist = SphereDist(sp, sptop->right);
          if(ldist < rdist){
            BubbleCheck(sp, sptop->left);
            BubbleCheck(sp, sptop->right);
          }else{
            BubbleCheck(sp, sptop->right);
            BubbleCheck(sp, sptop->left);
          }
        }
      }
```

【図３５】

Bubble Collision Check基本関数（その２）

```
}else if(sp->leaves->leafcnt > 1 && sptop->leaves->leafcnt == 1){ ・・・2106
    if(sp->left != 0 && sp->right == 0){
        BubbleCheck(sp->left, sptop);
    }else if(sp->left == 0 && sp->right != 0){
        BubbleCheck(sp->right, sptop);
    }else{
        ldist = SphereDist(sp->left, sptop);
        rdist = SphereDist(sp->right, sptop);
        if(ldist < rdist){
            BubbleCheck(sp->left, sptop);
            BubbleCheck(sp->right, sptop);
        }else{
            BubbleCheck(sp->right, sptop);
            BubbleCheck(sp->left, sptop);
        }
    }
}else if(sp->leaves->leafcnt == 1 && sptop->leaves->leafcnt > 1){ ・・・2107
    if(sptop->left != 0 && sptop->right == 0){
        BubbleCheck(sp, sptop->left);
    }else if(sptop->left == 0 && sptop->right != 0){
        BubbleCheck(sp, sptop->right);
    }else{
        ldist = SphereDist(sp, sptop->left);
        rdist = SphereDist(sp, sptop->right);
        if(ldist < rdist){
            BubbleCheck(sp, sptop->left);
            BubbleCheck(sp, sptop->right);
        }else{
            BubbleCheck(sp, sptop->right);
            BubbleCheck(sp, sptop->left);
        }
    }
}else{                                                          ・・・2108
    lf = sp->leaves->leaves;
    while(lp){
        lftop = sptop->leaves->leaves;
        while(lftop){
            if(Hash.Insert(lf, lftop) == TRUE){                 ・・・2109
                dist = lf->plane->CheckConvex(lftop->plane);    ・・2110
                if(dist < cq->dist){
                    cq->dist = dist * (1.0 - cq->alpha);        ・・2111
                }
            }
            lftop = lftop->next;
        }
        lf = lf->next;
    }
}
}
}
```

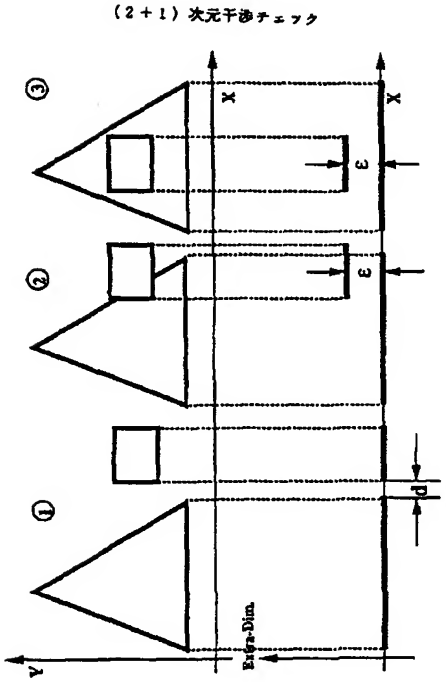
【図３８】

干渉チェックデータ構造（その２）

```
class MetaSphere{
  Vector3 center;
  Scalar r;
  MetaSphere *left;
  MetaSphere *right;
  MetaLeaves *metaleaves;
};

 class CollisionQue{
   ColQue *next;          // 第１ColQueリスト
   ColQue *next2;         // 第２ColQueリスト
   int colqnum;
   int colqnum2;
   MetaLeaf **MLfs;
   MetaLeaves **MLvleft;
   MetaLeaves **MLvright;
   MetaSphere **MSps;
   MetaLeaves *metaleaves;
   MetaSphere *metaspheres;
   MetaLeaf **MLfs2;
   MetaLeaves **MLvleft2;
   MetaLeaves **MLvright2;
   MetaSphere **MSps2;
   MetaLeaves *metaleaves2;
   MetaSphere *metaspheres2;
   Scalar Geps;  // 衝突判定用微小定数
   Scalar dist;  // Bubble Collision用距離判定変数
   Scalar dist0; // メタBubble Collision用距離判定変数
   Scalar alpha; // Bubble Collision用相対誤差
   int gcolflg;  // スイッチング用衝突フラグ
   ColQue *colqi;  // メタBubble Collision用
   ColQue *colpj;
   CollidePoint *cpoint;  // 多点接触表示用
   CollidePlane *cplane;  // 連続型Bubble Collision 用
 };
```

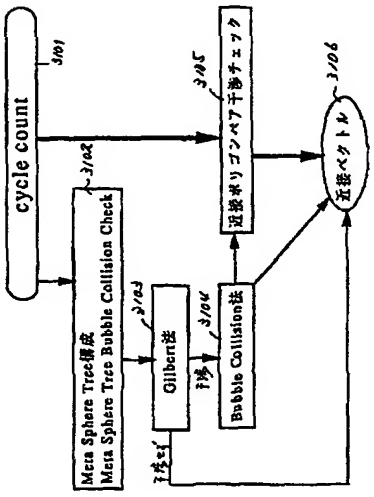
続く

【図４３】



【図４５】

メタツリー構成を包含した連続型干渉チェック
フロー図

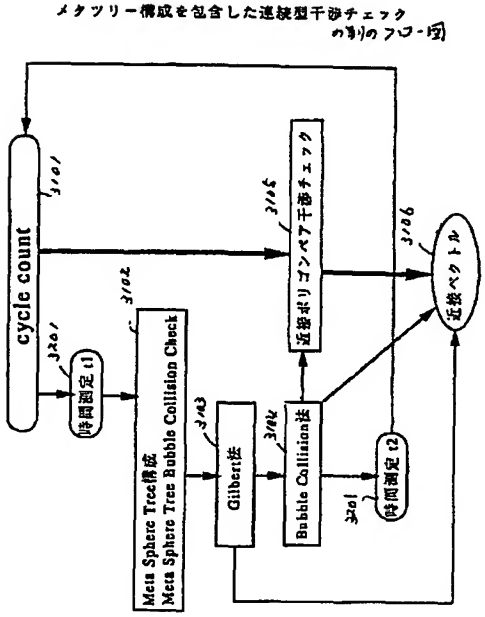


【図３９】

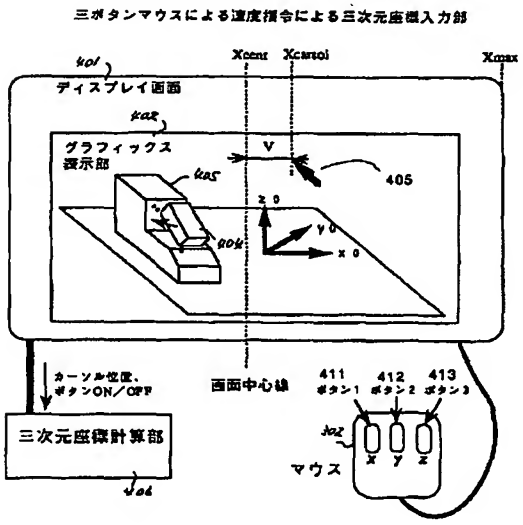
干渉チェックデータ構造（その3）

```
class Prims{
  Vector3 p;        // 頂点座標
  Vector3 n;        // 頂点法線
  Prims *next;      // 頂点リスト用ポインタ
  Prims *branch;    // 近傍頂点リスト用ポインタ
  int bcount;       // 近傍頂点個数
  int pnum;         // 頂点番号
};

class ConvexHull{
  Prims *top;       // 凸包頂点集合
  int count;        // 総頂点数
  Vector3 com;      // 重心
  Vector3 Pc;       // 凸包包絡球中心
  Scalar R;         // 凸包包絡球半径
  Vector3 max;      // 3次元各要素に対する最大、
  Vector3 min;      //   最小ベクトル
};
```

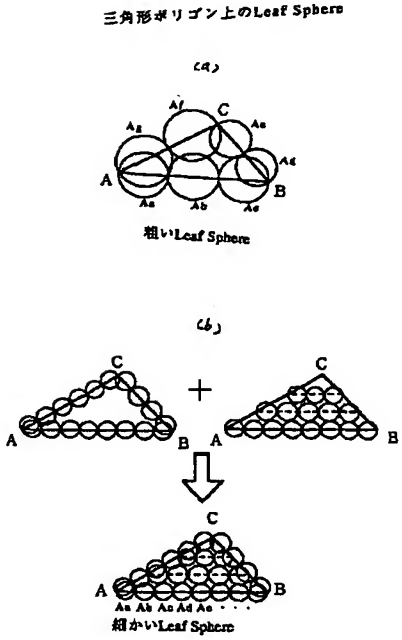
【図４６】



【図５０】

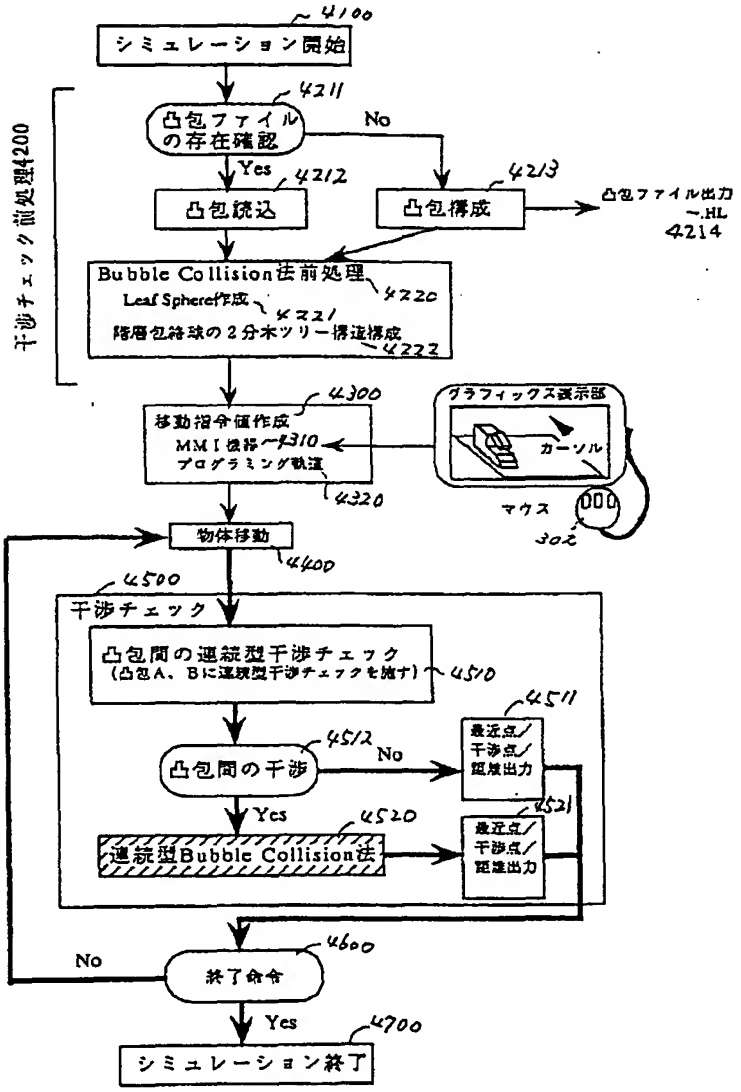


【図５２】



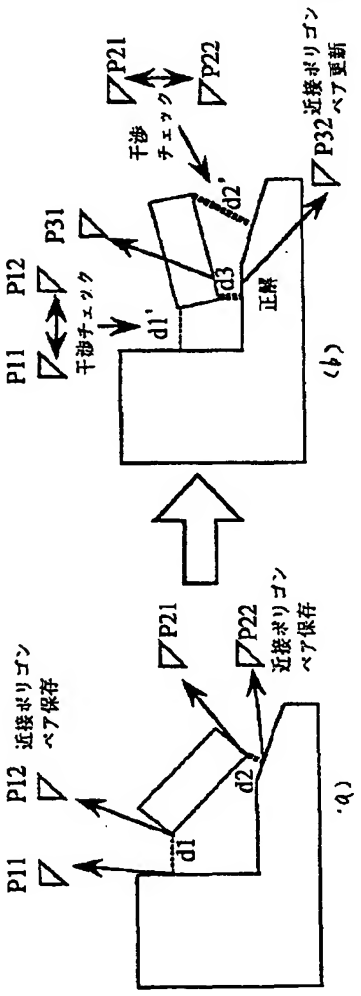
【図４７】

干渉チェックシステム全体フロー



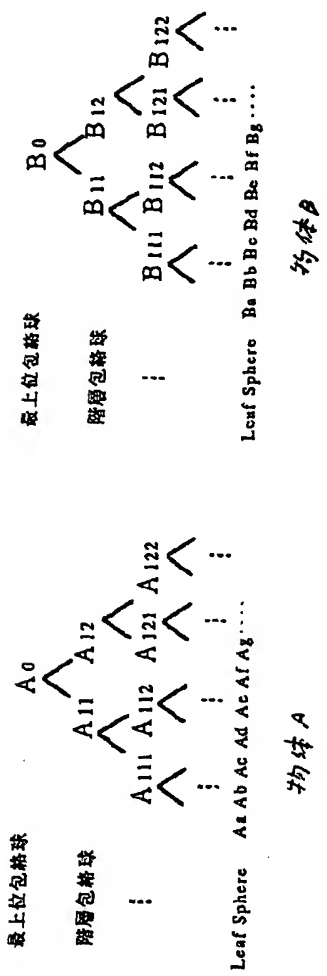
【図６１】

近接ポリゴンペア更新



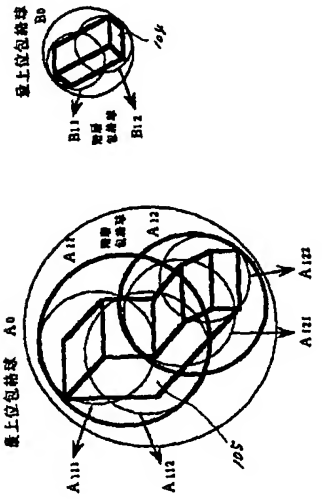
【図５３】

階層包絡球の２分木ツリー構造



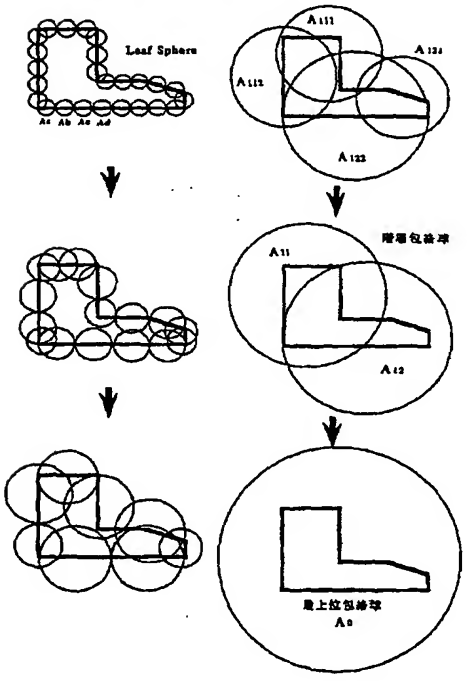
【図５４】

階層包絡球の２分木ツリー構造の３次元イメージ



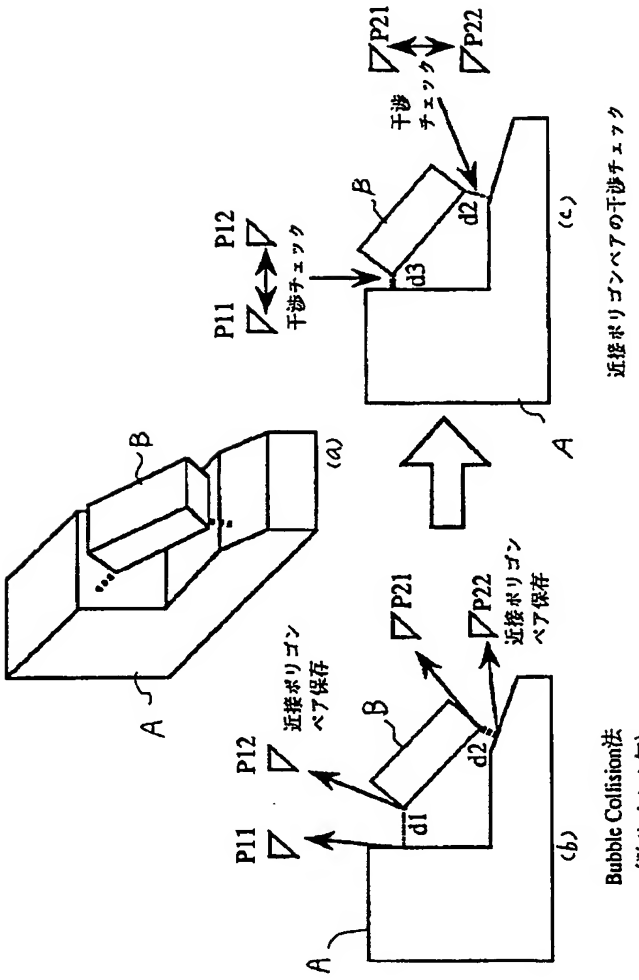
【図５６】

階層包絡球の構成例（非凸多面体Ａの場合）



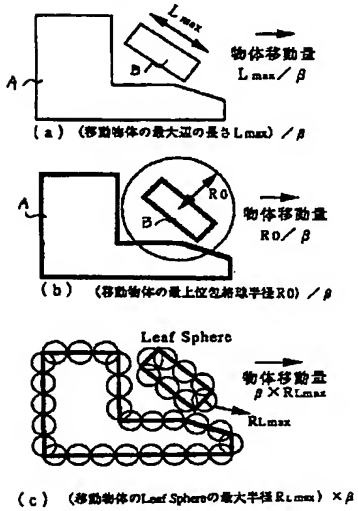
【図５７】

連続型Bubble Collision法



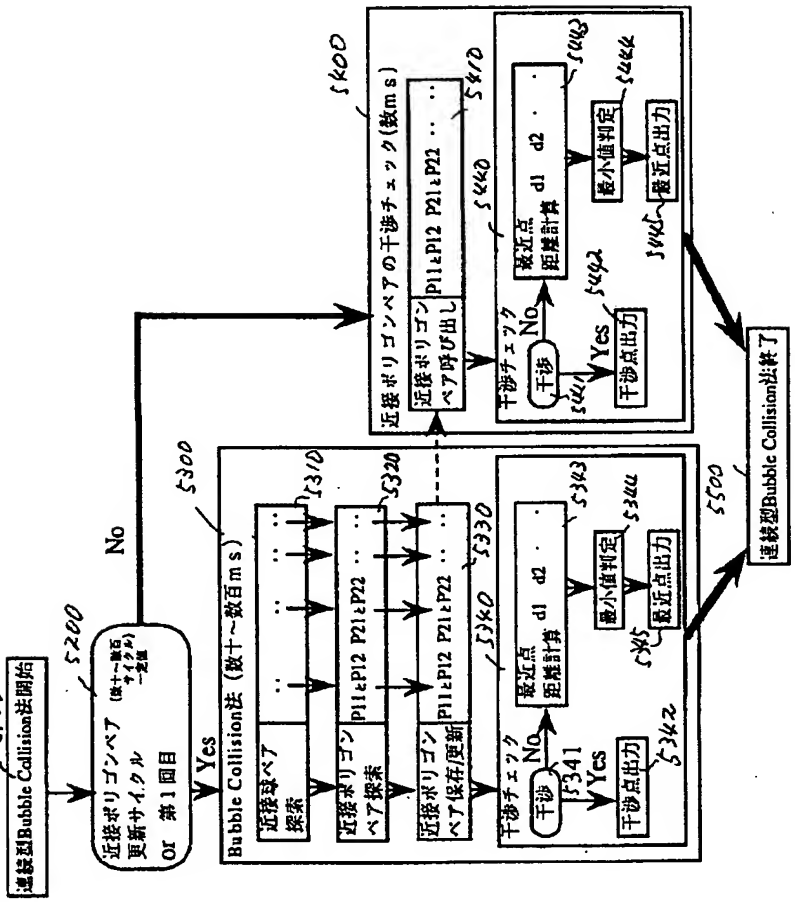
【図６５】

物体移動量のしきい値の決定法

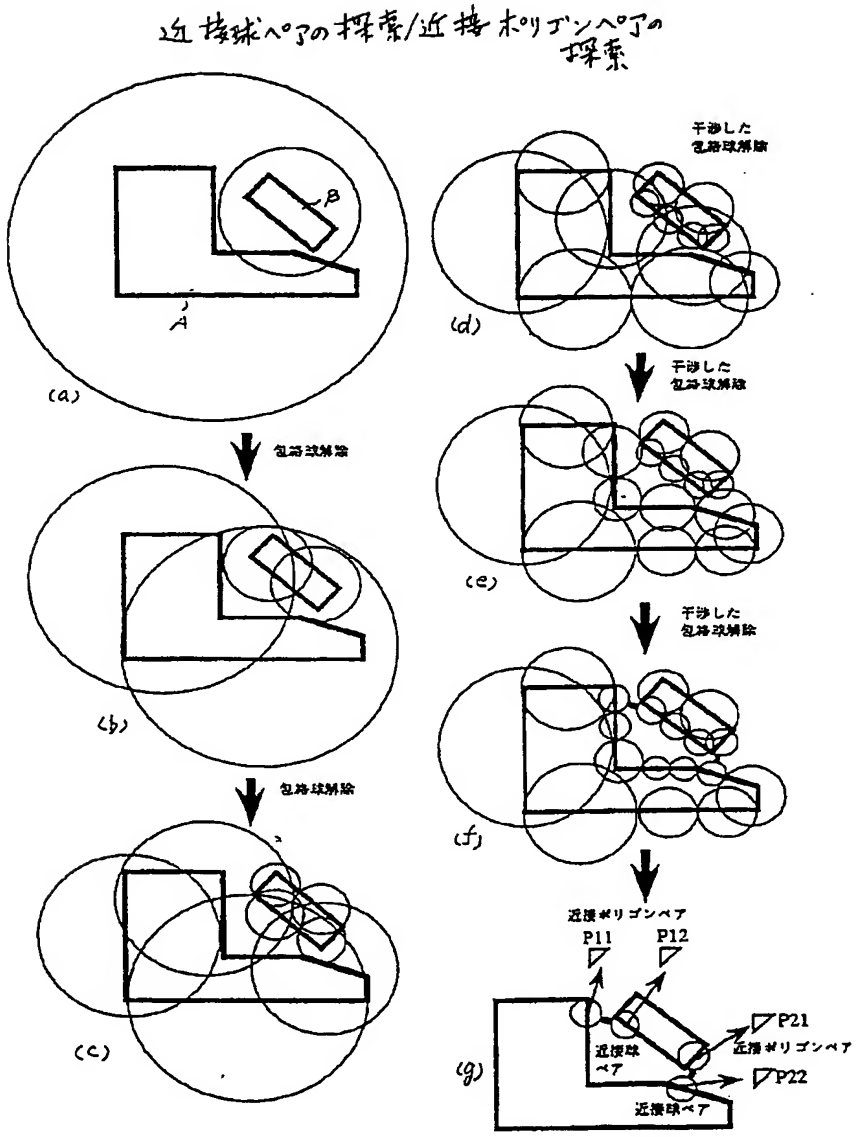


【図５８】

連続型Bubble Collision法のフロー

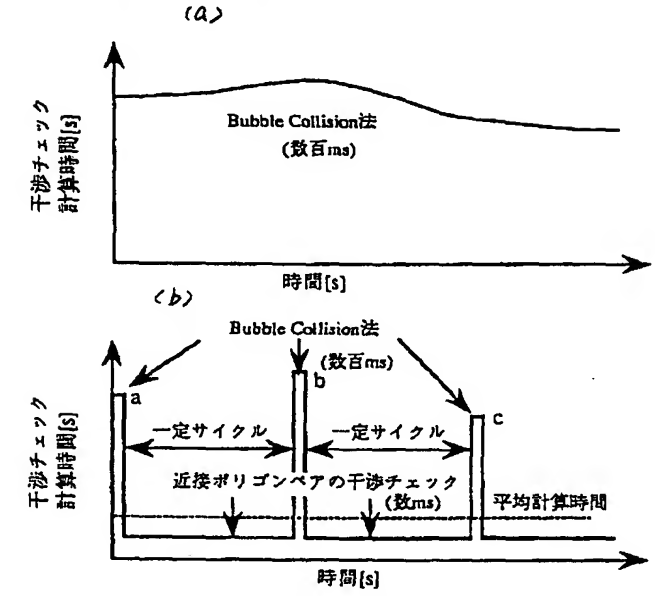


【図５９】


近接球ペアの探索/近接ポリゴンペアの探索
(a)
B
A
包絡球解除
(b)
包絡球解除
(c)
干渉した包絡球解除
(d)
干渉した包絡球解除
(e)
干渉した包絡球解除
(f)
近接ポリゴンペア
P11
P12
P21
近接ポリゴンペア
P22
近接球ペア
近接球ペア
(g)

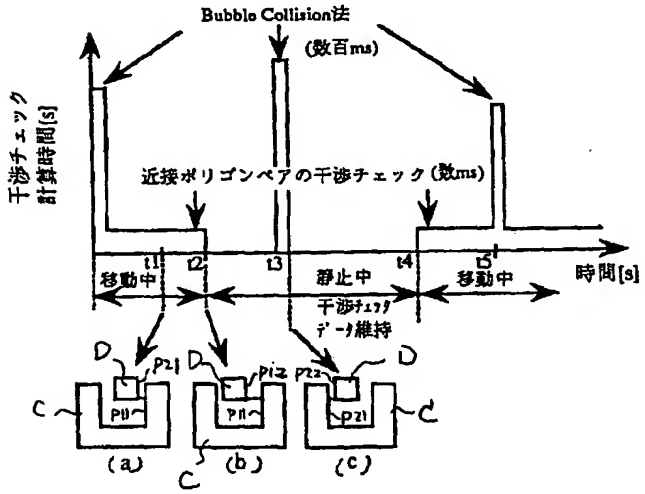
【図６０】

連続型Bubble Collision法の計算時間
（更新サイクル一定値）



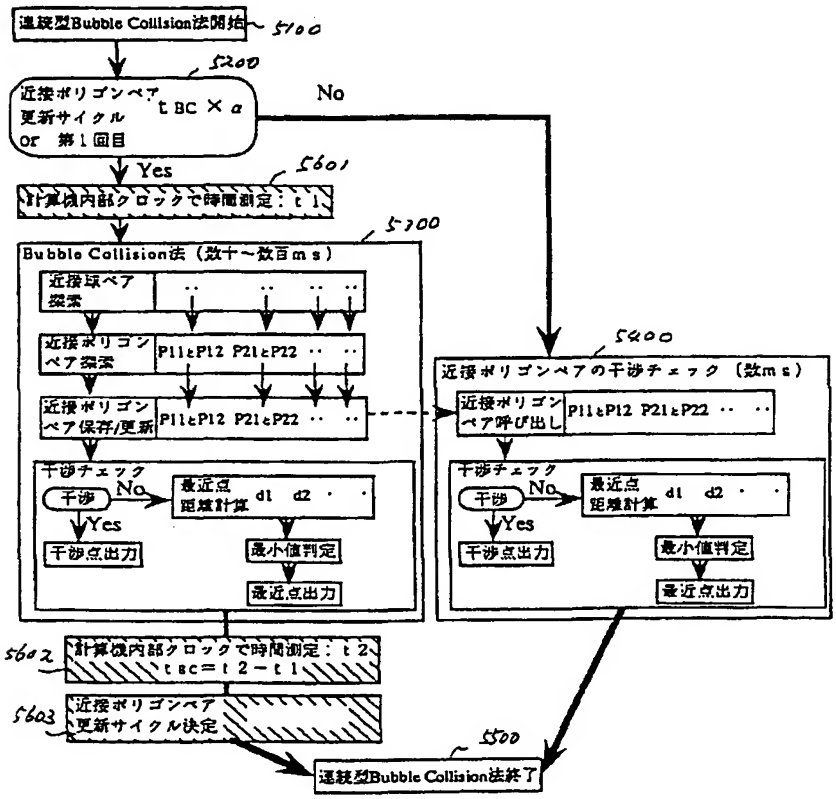
【図７０】

凸包内における物体静止時の干渉チェック効率化



【図６２】

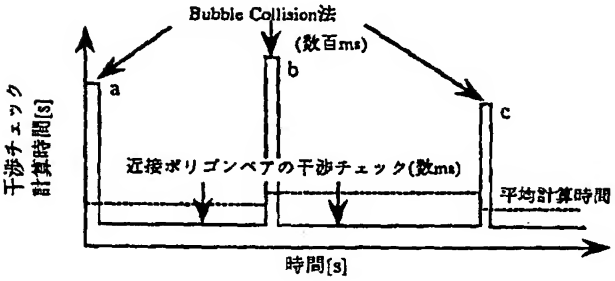
連続型Bubble Collision法の更新サイクル自動決定法のフロー



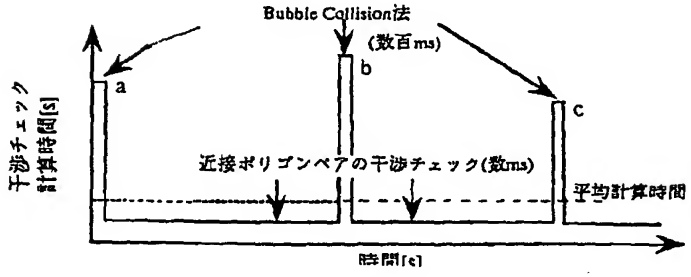
【図６３】

連続型Bubble Collision法の計算時間

(a)

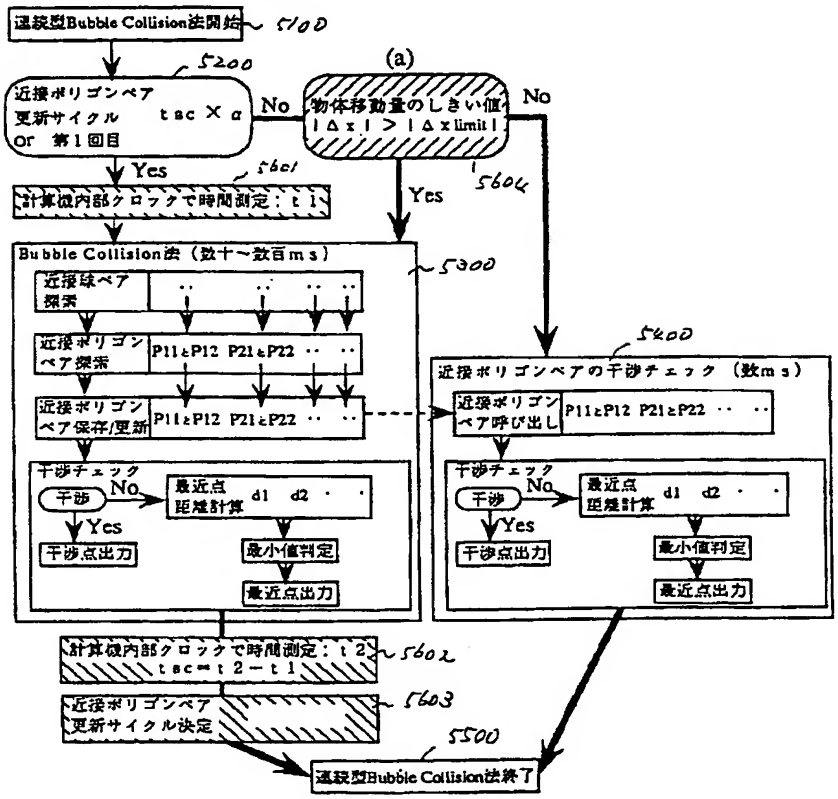


(b)



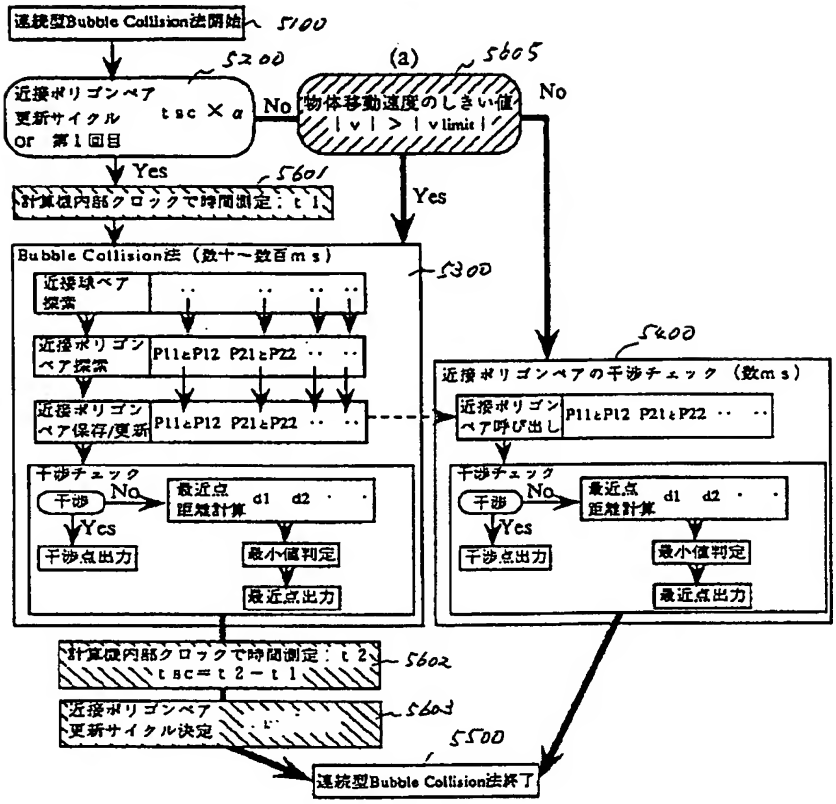
【図６４】

連続型Bubble Collision法の修正方法のフロー



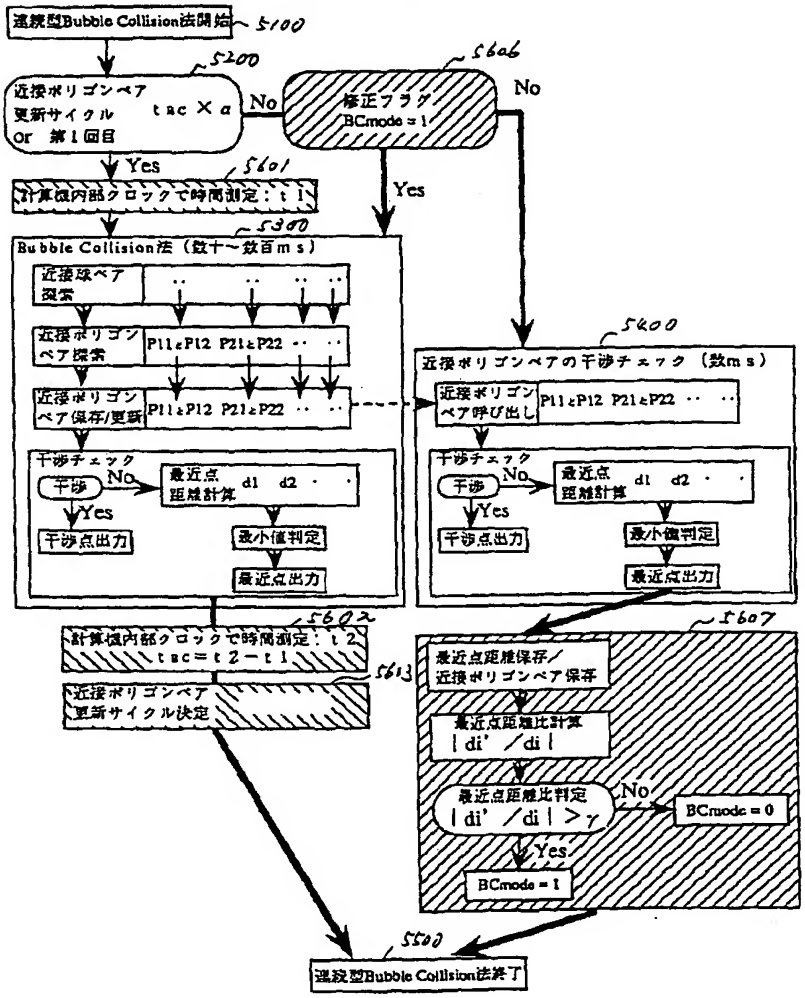
【図６６】

連続型Bubble Collision法の修正方法のフロー



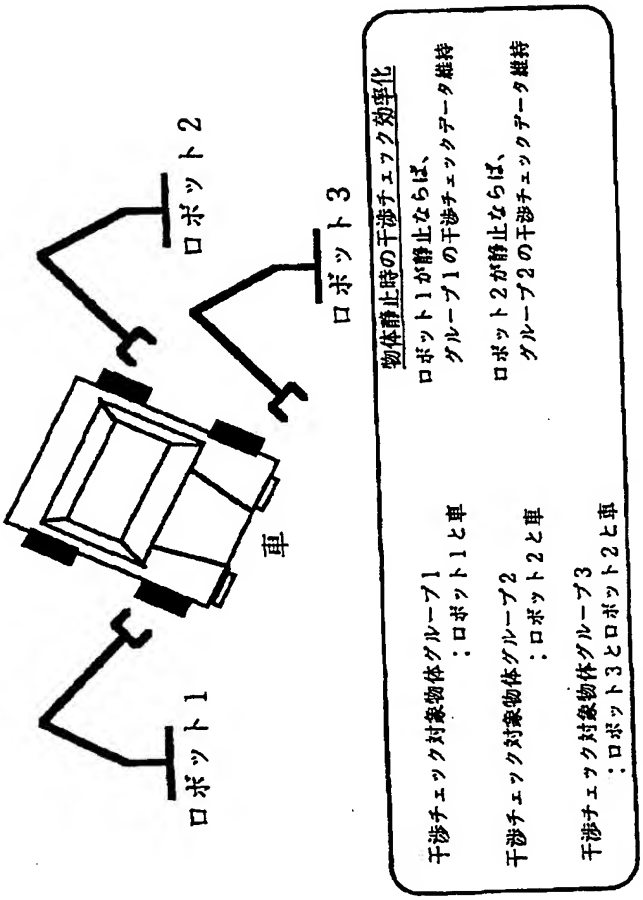
【図68】

連続型Bubble Collision法の修正方法のフロー



【図６９】

自動車工場内ロボットの干渉チェック



フロントページの続き

(56)参考文献　特開　平２－224004（ＪＰ，Ａ）
特開　平７－134735（ＪＰ，Ａ）

(58)調査した分野(Int.Cl.⁶，ＤＢ名)
G06F　17/50
G05B　19/19
G06T　17/00
ＪＩＣＳＴファイル（ＪＯＩＳ）